Matricola n. 0000758395

Alma Mater Studiorum

Università Di Bologna

Dipartimento Scienze Giuridiche

Corso Di Laurea Magistrale In Giurisprudenza

Uso del software libero e open source per l'analisi scientifica della prova digitale nell'informatica forense

Tesi di laurea in Informatica Forense

Relatrice:
Prof.ssa Raffaella Brighi

Presentata da:
Luca Piras

Sessione unica
Anno Accademico 2023/2024

Indice

Ringraziamenti

A Tyler e Jo

Desidero ringraziare i miei relatori, e tutti coloro che mi sono stati vicini durante la conclusione del mio percorso di studi, per il vostro supporto e la vostra pazienza.

Desidero anche ringraziare gli sviluppatori dei seguenti software liberi, perché sono stati usati per redigere la tesi:

- Emacs, Vim e Neovim – Per scrivere i file di testo.
- Git – Per la gestione dei file relativi alla tesi.
- Pandoc e TeX Live – Per la conversione della tesi in PDF.
- Okular – Per l'annotazione di file PDF.
- Borg, Restic e Rsync – Per la sincronizzazione di file, e creazione di copie di backup.
- Debian e Arch Linux – Sistemi operativi.
- Termux, Crostini e WSL – Possibilità di eseguire applicazioni Linux su Android, Chromebooks e Windows.

I font usati sono Libertinus Serif e Source Code Pro.

Introduzione

La materia e l'argomento di questa tesi sono stati scelti per un interesse personale per l'informatica, la programmazione, l'uso del software libero e dei sistemi operativi GNU/Linux.

L'obiettivo di questa tesi è di dimostrare che il software libero, o *open source*, è il modello di sviluppo e distribuzione del software che più di qualsiasi altro riesce a soddisfare le esigenze complesse dell'informatica forense.

Il primo capitolo definisce l'esatto oggetto dell'informatica forense e descrive brevemente le circostanze che hanno portato allo sviluppo della disciplina.

Segue l'esame dei problemi che rendono l'informatica forense una disciplina fragile:fragile: la rapidità con cui l'informatica si evolve, le difficoltà tecniche e legali che si incontrano nello studio scientifico dei sistemi, programmi e dati informatici, e la fragilità dei dati informatici.

Questi problemi comportano due conseguenze. La prima è l'applicazione degli stessi metodi e dello stesso rigore che vengono utilizzati nelle scienze naturali. La seconda è che la perizia è il mezzo di prova che deve essere preferito per il trattamento dei dati informatici.

Il secondo capitolo descrive le fasi del trattamento della prova, e le specifiche esigenze di natura processuale e scientifica per ciascuna di esse. In particolare, si giunge alla conclusione che la ricerca scientifica ed il processo sono animati da principi molto simili: il confronto fra le parti come il modo ideale per approssimare la realtà, la necessità di motivare le proprie conclusioni, la preferenza per la trasparenza invece

che la segretezza.

Dato che l'informatica forense è una disciplina che trasporta la ricerca scientifica all'interno del processo, segue che i principi appena indicati dovrebbero applicarsi anche all'informatica forense; e dato che l'informatica forense deve necessariamente usare programmi informatici per svolgere le proprie attività, allora è ragionevole pensare che anche gli strumenti usati dall'informatica debbano seguire gli stessi principi.

La seconda parte del secondo capitolo si concentra sull'inquadramento tecnico, legale e filosofico del software. La definizione di software libero e la distinzione fra codice sorgente e macchina diventeranno rilevanti nell'ultima parte del capitolo, ma è importante anticiparle. Segue una discussione del rapporto fra la legge sul diritto d'autore italiana ed il software, e come la l.d.a. in alcuni casi ostacola l'attività di ricerca scientifica, mentre in altri la legittima e protegge. La parte si conclude con una discussione dei contratti di licenza d'uso (che regolano i diritti che spettano a chi riceve una copia del software) e in particolare della licenza GPL (che mediante delle clausole particolari, riesce a garantire che il codice sorgente di un programma rimanga sempre disponibile ai suoi utilizzatori).

L'ultima parte del capitolo riprende le caratteristiche del software libero menzionate in precedenza e spiega la loro importanza. Si confronta il software libero con il software proprietario (non-libero) e si analizzano le conseguenze negative che derivano dall'uso di quest'ultimo, ed i vantaggi che invece deriverebbero dall'uso del software libero. Si conclude il capitolo discutendo l'impossibilità intrinseca di usare il software libero per sviluppare i captatori informatici.

Il terzo capitolo considera gli aspetti più pratici e tecnici dello sviluppo del software libero per l'informatica forense, ed è diviso in due parti.

Nella prima parte si indicano una serie di elementi di valutazione per argomentare che il software sia affidabile, come la scelta del linguaggio di programmazione, la documentazione, il corretto uso del codice di terze parti, l'uso di tecniche per garantire

che il software funzioni correttamente e il suo funzionamento sia riproducibile.

Nella seconda parte si elencano alcune buone pratiche relative allo sviluppo del software libero, per dimostrare che nonostante l'apertura al pubblico, il processo di sviluppo non è caotico, e che la trasparenza nel processo di sviluppo è preferibile alla segretezza.

L'ultimo capitolo elenca il software libero disponibile per il trattamento della prova nell'ambito dell'informatica forense. Si menziona l'importanza di usare un sistema operativo libero e si elenca una serie di software liberi che sono usati nella pratica per l'acquisizione, conservazione e analisi dei dati.

Capitolo 1

Approccio scientifico all'informatica forense e alla prova informatica

1.1 Definizione di informatica forense

Nel tempo, sono state date numerose definizioni dell'informatica forense,[1] che iniziano con "l'informatica forense riguarda la ...", e continuano con un elenco di attività tipiche.[2]

Queste definizioni presentano due problemi:

- Spesso si concentrano solo sugli aspetti tecnici e non sempre evidenziano l'aspetto "forense" della disciplina;[3]
- L'uso di elenchi dà l'impressione della tassatività[4] e pone problemi di

[1] Per un elenco, v. Antonio Gammarota, «Informatica forense e processo penale: la prova digitale tra innovazione normativa e incertezze giurisprudenziali», Alma Mater Studiorum – Università di Bologna, 2016, http://amsdottorato.unibo.it/7723/, p. 16.

[2] Ad esempio, la ricerca, protezione, identificazione, estrazione, documentazione, analisi, esibizione, conservazione, interpretazione, ecc. di mezzi di prova, prove informatiche o elettroniche, dati del computer, ecc.

[3] Ossia, il fatto che le attività dell'informatica forense sono tendenzialmente destinate a confluire all'interno di un procedimento giudiziario, e non sono semplicemente attività di studio, ricerca e analisi fini a sé stesse, o destinate ad essere utilizzate da privati.

[4] È difficile immaginare in anticipo ed in astratto ogni attività tipica, ed in ogni caso, l'evoluzione della disciplina può portare alla definizione di nuove attività tipiche.

interpretazione.[5]

Questi problemi possono essere superati usando definizioni che evidenziano il collegamento fra l'informatica ed il diritto, e fanno riferimento a concetti più generali ed astratti, invece di usare il metodo induttivo.[6]

Si consideri la definizione di *digital evidence*[7] data da Eoghan Casey[8]: "*[A]ny data stored or transmitted using a computer that support or refute a theory of how an offense occurred or that address critical elements of the offense such as intent or alibi.*"[9]

Il pregio di questa definizione è che collega esplicitamente la prova informatica al diritto penale, ed indica la sua funzione in maniera generale.[10] Il problema fondamentale è che non indica come la prova informatica deve essere trattata, e a quali discipline si deve fare riferimento.

La definizione di informatica forense data da A. Gammarota[11] risolve il secondo problema: "L'informatica forense studia le norme giuridiche ed le tecniche informatiche per il trattamento dei dati digitali a fini processuali." La definizione è breve, ma l'autore evidenzia la necessità di interpretarla nella maniera più ampia possibile:

- "Fini processuali": include qualsiasi procedimento decisionale in cui i dati informatici possano assumere rilevanza;[12]

[5]Ad esempio, si può discutere se i termini "conservazione" e "protezione" significhino la stessa cosa, oppure se la "conservazione" riguardi la creazione di copie di backup ed il controllo periodico dell'integrità dei dati, mentre la "protezione" riguardi l'applicazione di misure di sicurezza come cifratura e controllo degli accessi.

[6]Ossia, l'uso di una lista aperta di elementi, che si conclude con un'espressione come "e simili".

[7]Prova informatica.

[8]*Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*, USA, Academic Press, Inc., 2011, p. 7.

[9]Qualsiasi dato archiviato o trasmesso per mezzo di un computer che corrobora o confuta una teoria su come si sia verificato un reato, o che riguarda gli elementi più importanti del reato, come il movente o l'alibi.

[10]L'elenco al termine della definizione ha una finalità puramente illustrativa, e non definitoria.

[11]*Op. cit.*, pp. 26–27.

[12]Pertanto, non solo il processo e procedimento penali, ma anche quelli civili, amministrativi, tributari, ecc. Se le regole valgono per il processo che incide in maggiore misura sui diritti fondamentali della persona, a maggiore ragione devono valere per tutti gli altri tipi di processo.

- "Trattamento": significa la corretta gestione del dato informatico per qualsiasi fine,[13] e in qualsiasi momento;[14]
- "Norme giuridiche": significa qualsiasi norma che, anche se solo indirettamente o in maniera condizionata, può riguardare l'uso di dati informatici;
- "Tecniche informatiche": riguardano l'intera disciplina dell'informatica.[15]

1.2 Ambiti di rilevanza dell'informatica forense

1.2.1 Diritto penale sostanziale

L'impulso più forte allo sviluppo dell'informatica forense è stata la informatizzazione della società, e conseguentemente, l'uso di strumenti informatici per commettere reati. La particolarità degli strumenti informatici è che permettono anche ad un singolo individuo di causare danni enormi.[16]

Il problema fondamentale era la possibilità di reprimere queste nuove forme

[13]Le regole sul trattamento dei dati non valgono solo all'interno dei procedimenti decisionali, ma anche per garantire il corretto trattamento dei dati informatici come atti giuridici. Ad esempio, si consideri come l'uso di firme elettroniche fanno fede fino a prova contraria (art. 20 co. 1-ter CAD). L'uso e verifica delle firme digitali interessa all'informatica forense, anche se l'atto non sarà presumibilmente usato in un procedimento giudiziario.

[14]Anche prima dell'intervento del personale specializzato (ad esempio, la PG deve assicurare le fonti di prova; art. 55 co. 1 c.p.p.), e anche dopo la conclusione definitiva del procedimento (si pensi ai casi di impugnazioni straordinarie).

[15]Non solo il mondo accademico, ma anche l'attività di ricerca libera e non formalizzata svolta dall'industria, da ricercatori indipendenti, dai pratici della disciplina (v. A. Gammarota, *op. cit.*, p. 34), e le *best practices* (migliori prassi) che vengono pubblicate da enti nazionali, e da organizzazioni internazionali e sovranazionali (ad esempio, esistono una serie di standard ISO/IEC relativi al trattamento della prova informatica ed investigazione dei dati digitali, v. A. Gammarota, *ivi*, p. 27–28.

[16]Ad esempio, si pensi al *Morris worm*, che in 24 ore riuscì a diffondersi sul 10% dei 60.000 computer allora connessi a internet. L'obiettivo del *worm* non era di distruggere dati, ma di causare quello che oggi sarebbe chiamato un *denial of service attack* (attacco per l'interruzione del servizio). V. Federal Bureau of Investigation, *The Morris Worm*, 2018, https://www.fbi.gov/news/stories/morris-worm-30-years-since-first-major-attack-on-internet-110218. Ancora, si pensi alla distribuzione di materiale protetto dal diritto d'autore o di materiale pedo-pornografico mediante tecnologie *peer-to-peer* (v. Michele Ferrazzano, «Indagini forensi in tema di scambio di file pedopornografici mediante software di file sharing a mezzo peer-to-peer», Alma Mater Studiorum – Università di Bologna, 2014, http://amsdottorato.unibo.it/6697/, p. 99 ss.), dove una singola persona può distribuire quel materiale ad un numero potenzialmente illimitato di altri utenti.

di criminalità.[17] In molti casi era possibile qualificare il fatto commesso mediante un sistema informatico all'interno dei reati tradizionali.[18] Tuttavia, in alcuni casi questa operazione non era possibile[19] e gli stati iniziarono a dotarsi di leggi sui reati informatici.

Negli Stati Uniti, la prima legge fu approvata in Florida nel 1978 e quasi tutti gli altri Stati si dotarono di proprie leggi nel corso di poco più di un decennio.[20] In Italia, il codice penale viene riformato nel 1993.[21]

È possibile distinguere fra due tipi di reato, in base al bene giuridico protetto.[22]

Nei reati informatici "propri", il bene (o uno dei beni) protetti sono i dati o sistemi informatici.[23] Per la prova di questo tipo di reati, è assolutamente necessario acquisire i dati informatici.[24]

Nei reati informatici "impropri", i dati e sistemi informatici sono lo strumento con cui il reato è commesso, e non il bene protetto.[25] In questo caso, il reato non deve

[17]Il principio fondamentale del diritto penale è il principio di legalità, *nullum crimen sine lege* (nessun reato può esistere senza una legge).

[18]Ad esempio, negli Stati Uniti il fatto che i dati informatici avessero natura intangibile ed immateriale era irrilevante: che avevano comunque un valore economico, e pertanto si potevano applicare le figure di reato tradizionali; v. Hugh Nugent, «State Computer Crime Statutes», 1991, https://www.ojp.gov/ncjrs/virtual-library/abstracts/state-computer-crime-statutes, p. 2.

[19]Secondo alcune corti, questa interpretazione estensiva dei reati tradizionali sconfinava nella creazione di nuove figure di reato da parte dei giudici, che andava a violare i principi costituzionali del *due process of law* (giusto processo), o della separazione dei poteri. In altri casi, i reati commessi con strumenti informatici non potevano essere ricondotti alle figure di reato tradizionali, e pertanto era necessario crearne di nuove.

[20]H. Nugent, *op. cit.*, pp. 2–4.

[21]La Legge 23 dicembre 1993, n. 547 ("Modificazioni ed integrazioni alle norme del codice penale e del codice di procedura penale in tema di criminalità informatica.") modifica i reati tradizionali, aggiungendo riferimenti espliciti ai sistemi informatici e telematici, ed i dati e programmi in essi contenuti. V. A. Gammarota, *op. cit.*, p. 71.

[22]La distinzione è ripresa da A. Gammarota, *ivi*, p. 29.

[23]Ad esempio, nel danneggiamento di informazioni, dati e programmi informatici (art. 635-*bis* c.p.) l'unico bene protetto sono i dati in sé, mentre nella frode informatica (art. 640-*ter* c.p.) si protegge sia il corretto funzionamento del sistema informatico, sia il patrimonio della persona (ad esempio, i *ransomware* sono un caso di frode informatica, perché criptano i dati e programmi degli utenti, impedendone l'uso e accesso, e richiedono il pagamento di un riscatto per decriptarli).

[24]Altrimenti, sarebbe impossibile provare che il sistema informatico è stato aggredito dal reato.

[25]Ad esempio, si penis ad una truffa (art. 640 c.p.) dove gli "artifizi e raggiri" vengono creati mediante l'uso di sistemi informatici, come la clonazione della voce di una persona. V. A. Kohli, *From Scams to Music, AI Voice Cloning Is on the Rise*, 2023, https://web.archive.org/web/20230429203350/https://time.com/6275794/ai-voice-cloning-scams-music/.

essere necessariamente provato usando i dati informatici.[26]

Più in generale, i dati informatici possono essere utili per l'accertamento di qualsiasi altro reato.[27]

La definizione dei reati informatici getta le basi per lo sviluppo dell'informatica forense, perché per la loro prova può essere necessario, o comunque utile, trattare i dati informatici.[28] Allo stesso tempo, l'informatica forense è utile per meglio definire ed interpretare i reati informatici, ed i beni giuridici protetti.[29]

1.2.2 Diritto processuale penale

Il diritto processuale penale disciplina le modalità di trattamento delle prove[30] e dei mezzi di ricerca della prova.[31] Dato che l'introduzione dei reati informatici è un fenomeno estremamente recente, la cultura processuale ha dovuto adattare gli istituti tradizionali, pensati per prove materiali, ai dati informatici, che invece hanno una natura e caratteristiche diverse.

Negli Stati Uniti la cultura processuale è stata sempre aperta alla discussione dei principi, strumenti e metodologie scientifiche usate per ricostruire e valutare il fatto all'interno del contraddittorio, e questo adattamento non ha incontrato difficoltà. Viceversa, in Italia la cultura processuale preferisce l'uso delle prove precostituite alle

[26] Perlomeno, almeno in teoria non è strettamente necessario doverli acquisire per poter provare il fatto, ma naturalmente, la ricostruzione del fatto sarebbe agevolata avendo a disposizione più informazioni possibile.

[27] Ad esempio, si pensi al caso di un reato commesso da più persone, che usano gli strumenti informatici al solo fine di comunicare fra di loro. Il reato può essere sicuramente provato in altri modi, ma queste informazioni sono comunque utili per dimostrare la responsabilità dei vari soggetti.

[28] È preferibile usare sempre la massima cautela nel trattamento dei dati, anche se si pensa di usarli solo come elementi per orientare le indagini, perché i dati informatici sono estremamente fragili, e possono essere modificati o cancellati con facilità.

[29] L'informatica forense aiuta il legislatore a rispettare il principio di tassatività, per cui i reati devono essere descritti in maniera chiara e precisa, specie se fanno riferimento a nozioni tecniche, ed il principio di offensività, per cui i reati devono proteggere i beni giuridici solo dalle modalità di aggressione più gravi.

[30] Le modalità con cui le prove devono essere assunte (che devono rispettare i diritti fondamentali della persona), e le sanzioni processuali nel caso in cui queste modalità vengano violate.

[31] Chi può ricercare le prove, secondo quali modalità e limiti, le sanzioni nel caso di violazione della disciplina.

prove costituende.[32]

La preferenza per le prove precostituite pone due rischi. Il legislatore non può prevedere regole e cautele particolari per la loro assunzione, ed eventuali sanzioni per la violazione di queste regole,[33] e il giudice potrebbe ritenere che non sia necessario l'intervento di un perito o consulenti tecnici per la corretta assunzione e valutazione della prova.[34]

La legge di ratifica[35] della Convezione di Budapest del 2001[36] ha cercato di porre rimedio a questa situazione introducendo dei principi per il trattamento dei dati informatici all'interno del codice di procedura.

Tuttavia, questo intervento può essere criticato, perché è limitato solo ad alcuni istituti (principalmente i mezzi di ricerca della prova)[37] e non sono previste sanzioni processuali per la violazione di quei principi.[38]

---

[32]Ossia, si preferiscono le prove di natura documentale, che si formano al di fuori del processo, rispetto alle prove che si formano all'interno del dibattimento. V A. Gammarota, *op. cit.*, pp. 11–12, 22–23

[33]Ad esempio, si consideri la disciplina minuziosa prevista per la ricognizione di persone. Un intero articolo è dedicato agli atti preliminari, si prevede l'esecuzione e la menzione di questi adempimenti nel verbale a pena di nullità, e gli articoli successivi regolano lo svolgimento, e altri tipi di ricognizione (artt. 213–216 c.p.p.). Il legislatore sa che la memoria umana è labile, e quindi istituisce una disciplina articolata per una prova che si fonda interamente sul riconoscere qualcuno o qualcosa, a distanza di tempo.

[34]Ad esempio, si potrebbe argomentare che un documento in formato PDF non sia troppo diverso da un documento cartaceo, e quindi che possa essere acquisito e valutato come qualsiasi altro documento. Tuttavia, questa ricostruzione è eccessivamente semplicistica, perché ignora le problematiche tecniche relative alla corretta copia di un documento informatico, alla presenza di modifiche che non sono immediatamente apparenti, etc.

[35]Legge 18 marzo 2008, n. 48, "Ratifica ed esecuzione della Convenzione del Consiglio d'Europa sulla criminalità informatica, fatta a Budapest il 23 novembre 2001, e norme di adeguamento dell'ordinamento interno."

[36]Convention on Cybercrime (ETS No. 185), v. https://rm.coe.int/1680081561.

[37]Le modifiche riguardano le ispezioni (art. 244), perquisizioni (art. 247), il sequestro di corrispondenza (art. 254 e 254-bis), la custodia delle cose sequestrate (art. 259), le perquisizioni in caso di flagranza o evasione (art. 352) e gli accertamenti urgenti (art. 354). Il legislatore evidenzia la necessità di conservare i dati originali, impedire la loro modificazione, e copiarli con procedure che garantiscano che la copia dei dati sia identica all'originale. È importante disporre le massime garanzie nella fase iniziale, perché se i dati sono acquisiti incorrettamente, tutte le fasi successive saranno viziate a loro volta, e ancora peggio, potrebbe essere impossibile acquisire i dati di nuovo, perché sono stati modificati. Tuttavia, sarebbe stato opportuno prevedere dei principi anche per gli istituti relativi alle prove, come ad esempio, un divieto di acquisire documenti informatici, se non si dimostra che sono stati acquisiti secondo gli stessi principi previsti per il loro sequestro (art. 254-*bis* c.p.p.).

[38]Pertanto, si corre il rischio che la fase del dibattimento venga contaminata da prove assunte

1.2.3 Altre branche del diritto

I principi, le tecniche e le conoscenze dell'informatica forense possono trovare applicazione anche nelle altre branche del diritto, ogni volta che il trattamento dei dati informatici diventa rilevante per il compimento di una qualsiasi operazione che produce effetti giuridici,[39] come la dimostrazione della responsabilità,[40] l'applicazione di sanzioni[41] e l'adozione di atti che producono effetti giuridici.[42] Inoltre, devono anche essere applicate agli atti giuridici in formato digitale.[43]

1.3 Problemi dell'informatica forense

L'informatica forense è una disciplina che presenta una serie di caratteristiche particolari, che possono farla apparire instabile o fragile.

Un primo problema è la continua e rapida evoluzione delle tecnologie informatiche, sia a livello di hardware che di software.[44] Pertanto, i metodi di analisi tendono ad essere

incorrettamente. Il giudice dovrebbe dichiarare quelle prove se non inammissibili (per mancanza di una sanzione espressa), almeno inaffidabili, ma la sua valutazione delle altre prove potrebbe essere comunque falsata dalla presenza delle prove informatiche.

[39] È irragionevole pensare che gli standard stringenti dell'informatica forense debbano trovare applicazione solo all'interno del diritto penale, perché è la branca del diritto che più di tutte va a incidere sui diritti fondamentali della persona. L'informatica forense prevede quegli standard perché i dati informatici hanno delle caratteristiche intrinseche particolari che li necessitano, e pertanto, devono essere sempre trattati allo stesso modo, indipendentemente dal contesto in cui vengono usati.

[40] Ad esempio, si immagini l'uso di dati informatici come prove in un giudizio civile, amministrativo, tributario, ecc.; oppure nelle procedure di *ADR* (*alternative dispute resolution*, risoluzione alternativa delle controversie) come l'arbitrato

[41] Ad esempio, i procedimenti disciplinari nei confronti di dipendenti della PA, avvocati, sportivi, ecc.

[42] Ad esempio, si pensi all'istruttoria nel procedimento amministrativo, finalizzata all'emanazione di un atto. È importante garantire almeno la corretta acquisizione e conservazione dei dati informatici che saranno usati ai fini della decisione.

[43] Ad esempio, nel processo telematico gli atti stessi del processo sono rappresentati come dati informatici, ed è estremamente importante dimostrare l'autenticità mediante l'uso di firme digitali, garantire la loro corretta conservazione, ed essere in grado di rilevare eventuali modificazioni. Lo stesso ragionamento vale anche per qualsiasi atto con valore giuridico che si forma e ha rilevanza all'esterno del processo, specie se quell'atto potrebbe formare in seguito oggetto di controversia (ad esempio, atti della PA, o contratti fra privati). Maggiori sono le cautele utilizzate, e maggiore è l'affidabilità di quell'atto.

[44] Nell'ambito delle scienze naturali, i fenomeni naturali non cambiano, cambiano solo le teorie ed i modelli che gli scienziati sviluppano per spiegarli. Viceversa, nell'ambito dell'informatica forense, i supporti, sistemi, programmi e dati informatici sono in continua evoluzione.

sempre innovativi, e nel tempo necessario per svolgere una *peer review* approfondita, potrebbero già essere diventati obsoleti.[45]

Un altro problema è che l'attività di ricerca non è libera, ma è vincolata da vari tipi di limiti: materiali,[46] tecnici,[47] relativi alla documentazione,[48] e legali.[49]

L'ultimo problema è la fragilità dei dati informatici: è difficile prevenire, rilevare o annullare la loro dispersione o modifica, che sia accidentale o intenzionale.

Per quanto riguarda le modifiche accidentali, la conservazione e trasmissione dei dati richiede sempre una modifica della realtà materiale,[50] e pertanto,

[45]Il problema è aggravato dal fatto che il sistema operativo ed il software vengono spesso aggiornati in maniera automatica all'ultima versione, senza l'intervento dell'utente, per esigenze di sicurezza informatica. Pertanto, anche se i dati che sono stati acquisiti in un certo momento non varieranno nel tempo, la ricerca scientifica potrebbe non avere ancora a disposizione strumenti comprovati e maturi per analizzarli.

[46]I supporti materiali su cui i dati sono memorizzati potrebbero non essere rimovibili (è il classico caso degli smartphone, o di altri dispositivi creati per uno scopo particolare, come gli apparecchi elettromedicali) o potrebbero usare interfacce proprietarie. In entrambi i casi, l'estrazione dei dati dal dispositivo potrebbe essere impossibile o fortemente limitata.

[47]Anche laddove sia possibile acquisire tutti i dati da un dispositivo, è possibile che il software usato da quel dispositivo usi misure di protezione tecniche per rendere più difficile la sua analisi. Ad esempio, tecniche come la crittografia, *code obfuscation*, ecc., che complicano lo studio del funzionamento del software.

[48]Il funzionamento dei programmi ed il formato dei dati spesso non sono documentati pubblicamente. La documentazione potrebbe essere fornita solo a sviluppatori di terze parti, ma solo a discrezione degli sviluppatori originali, e comunque con un vincolo di *NDA* (*non-disclosure agreement*, accordo di riservatezza), in modo che queste informazioni non possano essere condivise con il pubblico. Tuttavia, anche se questa documentazione viene fornita, potrebbe essere incompleta o inadeguata per i fini specifici delle investigazioni di informatica forense, che mirano ad accertare l'integrità (non-manomissione) dei dati, e ricostruire le dinamiche che hanno portato ad un certo assetto dei dati. È molto probabile che le specifiche tecniche dei file e protocolli siano documentati in dettaglio, ma è meno probabile che il funzionamento del sistema operativo e dei programmi (quali file aprono, modificano, creano od eliminano durante il loro funzionamento, come i file vengono modificati, a quali condizioni, in che formato, ecc.) siano oggetto di descrizione. Pertanto devono essere ricostruiti con un lungo, dispendioso, e potenzialmente inaffidabile processo di tentativi ed errori.

[49]In ogni caso, istituti come il diritto d'autore, la disciplina relativa ai brevetti e ai segreti industriali, etc. possono porre dei limiti all'attività di ricerca scientifica. Ad esempio, possono limitare la possibilità di creare copie del software oggetto di studio, di rimuovere mezzi di protezione per studiare meglio il software, di re-implementare il funzionamento del software per creare uno strumento di analisi, possono giustificare il rifiuto di fornire informazioni sul funzionamento del software, ecc.

[50]La modifica può anche essere di breve durata, o riguardare un'area limitata, ma deve comunque essere misurabile. Se così non fosse, i dati digitali sarebbero completamente immateriali e immaginari. Ad esempio, le modalità di trasmissione senza fili (Bluetooth, Wi-Fi, ecc.) hanno un raggio utile limitato, al di fuori del quale la trasmissione diventa impossibile, ed i dati che vengono trasmessi smettono di esistere se la trasmissione viene interrotta. Ancora, i dati nella memoria RAM sono memorizzati, non trasmessi, ma si disperdono appena il sistema viene spento.

qualsiasi problema nella realtà materiale si riflette sui dati. Il deterioramento[51] del supporto materiale comporta la graduale perdita di funzionalità del supporto.[52] Il malfunzionamento[53] può dipendere da cause "naturali"[54] o cause "meccaniche".[55]

Per quanto riguarda le modifiche intenzionali, in linea teorica, se è possibile entrare in possesso del supporto, è anche possibile modificare i suoi contenuti. La distruzione integrale di tutti i dati mediante sovrascrittura è considerata irreversibile.[56] La modifica arbitraria di contenuti specifici del disco[57] è più complessa, e può essere rilevata mediante l'uso di varie tecniche.[58]

[51]Inteso come il processo naturale, inevitabile ed irreversibile per cui tutta la materia tende progressivamente verso il disordine.

[52]Nell'ipotesi migliore, il sistema rileva la presenza di settori corrotti che sono diventati illeggibili, lo comunica all'utente, ed in alcuni casi, cerca di ripristinare i dati. Ad esempio, se si hanno due dischi configurati in modo da usare lo schema di archiviazione RAID 1, entrambi i dischi contengono una copia identica dei dati. Se un settore è corrotto su un disco, è possibile recuperare i dati dall'altro disco. Nei casi più gravi, il sistema non si accorge che un settore è corrotto, e restituisce un dato errato senza informare l'utente. Nel caso peggiore, l'intero supporto non viene più riconosciuto dal sistema. A quel punto, è necessario utilizzare tecniche particolarmente invasive per cercare di recuperare i dati, che richiedono lo smontaggio irreversibile del supporto materiale, ed in ogni caso, pongono problemi dal punto di vista della loro affidabilità.

[53]Inteso come un fenomeno estremamente raro da un punto di vista statistico, per cui un supporto non deteriorato si comporta in maniera erronea.

[54]Come i *bit flip* (inversione di singoli bit) dovuta a raggi cosmici. V. T. Long, *This Week in Glean: What Flips Your Bit?*, 2022, https://web.archive.org/web/20220413132337/https://blog.mozilla.org/data/2022/04/13/this-week-in-glean-what-flips-your-bit/.

[55]Come gli *unrecoverable read errors* (errori di lettura irrimediabili) che sono dovuti al fatto che il supporto materiale è pur sempre un oggetto imperfetto, che può occasionalmente compiere errori. V. T. Pott, I. Thomson, *Flash banishes the spectre of the unrecoverable data error*, 2015, https://web.archive.org/web/20200707202632/https://www.theregister.com/2015/05/07/flash_banishes_the_spectre_of_the_unrecoverable_data_error/.

[56]V. Daniel Feenberg, «Can Intelligence Agencies Read Overwritten Data?», 2013, https://back.nber.org/sys-admin/overwritten-data-guttman.html. Tradizionalmente si raccomandava l'uso di numerosi passaggi (v. Peter Gutmann, «Secure Deletion of Data from Magnetic and Solid-State Memory», 1996, https://www.usenix.org/legacy/publications/library/proceedings/sec96/full_papers/gutmann/index.html), ma successivamente è stato dimostrato che è sufficiente un singolo passaggio, che imposta tutti i bit a zero (v. Miles Wolbe, «Can data be recovered from a zero-filled hard drive?», 2018, https://tinyapps.org/docs/recovering_data_from_zero_filled_hard_drive.html).

[57]Ad esempio, eliminare solo alcuni file, o modificare i loro contenuti.

[58]La semplice eliminazione di un file non rimuove immediatamente i suoi contenuti, ma li segna solo come spazio libero. Usando software specializzati (ad esempio, *PhotoRec*), è possibile esaminare le aree del supporto segnate per ricercare file cancellati. Se il file è stato sovrascritto prima di essere eliminato (ad esempio, con GNU *shred*, v. Free Software Foundation, «GNU Coreutils», 2023, https://web.archive.org/web/20240205001115/https://www.gnu.org/software/coreutils/manual/html_node/index.html, sez. 11.6), è possibile che una copia dei contenuti del file possa essere trovata altrove. Ad esempio, se il file è un'immagine o un video, il sistema operativo spesso produce una *thumbnail* (anteprima) dei contenuti di quel file. Sovrascrivere il file non elimina automaticamente anche l'anteprima, che è salvata

Per cercare di impedire questo tipo di modifiche, si possono usare delle misure di sicurezza, che possono essere ricondotte a due grandi famiglie. Le misure di sicurezza software[59] sono efficaci solo quando il sistema è attivo[60] e possono essere aggirate con relativa facilità.[61] Le misure di sicurezza hardware[62] sono sempre attive, e se implementate correttamente, sono pressoché impossibili da aggirare.[63]

Un'altra caratteristica dei dati informatici è il fatto che le modifiche non lasciano tracce. Nel caso in cui le misure di sicurezza vengono aggirate, ed i dati informatici vengono modificati, è impossibile risalire con certezza al loro stato precedente, e tutte le ricostruzioni sono al più ipotesi.[64]

1.4 Rigore scientifico nell'informatica forense

Date queste premesse, potrebbe sembrare che l'informatica forense sia una disciplina fragile ed instabile, tendenzialmente incapace di fornire elementi utili

in maniera indipendente rispetto al file. Per quanto riguarda la modifica dei file, è necessario verificare se il sistema operativo o altre applicazioni tengono traccia dell'integrità dei file o delle operazioni compiute mediante *checksum* o *log files*. In questi casi, è possibile confrontare se il file corrisponde o meno a quanto ci si aspetta sulla base di questi valori di riferimento.

[59] Ad esempio, il software che normalmente richiede la password per accedere all'account di un utente, o impedisce che l'utente attualmente autenticato possa visualizzare o modificare file di altri utenti, o file gestiti dal sistema operativo.

[60] Se il sistema è spento, non sono in esecuzione. L'unica eccezione è la *encryption-at-rest* (crittografia a riposo), dove i dati rimangono criptati (e quindi illeggibili a chiunque non conosca la chiave per decrittarli) anche quando il sistema è spento.

[61] Spesso il software presenta degli errori di programmazione che possono essere oggetto di *exploit* (sfruttati) per aggirare le misure di sicurezza.

[62] Ad esempio, il supporto si rifiuta di funzionare a meno che non venga inserita una password mediante dei pulsanti fisici, un *USB dongle*, etc.

[63] Ad esempio, i dispositivi prodotti dalla Apple negli ultimi anni includono varie misure di sicurezza a livello hardware, che rendono difficile manomettere il sistema operativo, o decrittare i dati dell'utente. V. Apple Inc., «Apple Platform Security», 2022, https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf, p. 7.

[64] Nel mondo materiale, è praticamente impossibile agire senza lasciare una qualche minima traccia, ma nel mondo digitale, i singoli bit sono già l'unità di memoria minima. Ad esempio, sovrascrivere un singolo bit trasforma la sequenza di caratteri "1966" in "1946" (v. A. Gammarota, *op. cit.*, p. 62), ed è praticamente impossibile trovare tracce di questo cambiamento. I bit adiacenti sono rimasti inalterati, e il bit che è stato sovrascritto non mantiene traccia del suo valore precedente. Se quella data non occorre altrove, la modifica non può essere rilevata, e anche se occorresse altrove, sorgerebbe il problema di spiegare le incongruenze e capire quale sia la data autentica.

all'interno di un processo. Tuttavia, è proprio questa fragilità che giustifica (se non impone) l'uso di un approccio scientifico e rigoroso.

L'informatica forense si appoggia all'informatica, che non è né una scienza sociale[65] né una scienza naturale.[66] Pertanto, si potrebbe dubitare della natura scientifica dell'informatica forense.[67]

L'informatica è una disciplina di natura puramente teorica, ed è analoga alla matematica: sia la matematica, sia l'informatica usano concetti astratti,[68] che vengono combinati fra di loro per mezzo di ragionamenti deduttivi[69] e algoritmi.[70] Entrambe sono strumenti usati dalle altre scienze, ma non sono scienze in sé,[71] data la loro natura deduttiva. Tuttavia, è possibile argomentare che l'informatica forense[72] sia una scienza

[65]L'informatica può essere usata dalle scienze sociali per analizzare grandi quantità di dati, ma interessa principalmente come strumento, non come scienza. Si pensi al settore della *data analytics*, che analizza enormi quantità di dati grezzi relativi ad utenti di siti internet per studiare, ed eventualmente manipolare, fenomeni sociali (ad esempio, a quali temi gli utenti sono interessati, e come incentivarli ad acquistare determinati prodotti), e la relativa regolazione da parte del diritto, che cerca di regolare l'utilizzazione dei dati personali resi disponibili dagli utenti su internet, con strumenti come la GDPR.

[66]L'oggetto dello studio dell'informatica è l'elaborazione automatica delle informazioni, e "cosa si può fare con l'informazione", e non l'informazione come "fenomeno naturale", e "come l'informazione esiste", che invece è oggetto di studio di altre discipline, come la fisica.

[67]L'aggettivo "forense" nell'espressione "informatica forense" suggerisce la sua affinità con le altre scienze forensi, che possono essere legate sia alle scienze naturali (ad esempio, la medicina legale e la tossicologia forense sono strettamente legate alla biologia, la balistica forense è legata alla fisica), o alle scienze sociali (ad esempio, la criminologia e la psicologia).

[68]Ad esempio, la matematica usa concetti come "numeri", "punti", "rette", e l'informatica usa concetti come "bit" e "byte", gli operatori booleani, le funzioni. In entrambi i casi, le discipline danno solo una definizione assiomatica di questi concetti, e non si preoccupano di studiare come rappresentarli nella realtà materiale.

[69]Un teorema è un ragionamento logico che è sempre valido, e prescinde dall'uso di osservazioni empiriche per essere dimostrato. Ad esempio, gli *Elementi* di Euclide, ed il *lambda calculus* di Church sono esempi di sistemi di regole create con il metodo deduttivo. Viceversa, le scienze naturali usano un modello induttivo, e creano un sistema di regole sulla base dell'osservazione di numerosi fenomeni empirici.

[70]Gli algoritmi sono sequenze di istruzioni che non hanno un contenuto definitorio o conoscitivo, ma solamente imperativo. Un algoritmo spiega come manipolare delle informazioni per arrivare ad un certo risultato. Le scienze naturali cercano di definire e spiegare il funzionamento dei fenomeni naturali.

[71]V. Stefan Bilaniuk, «Is mathematics a science?», 1996, http://euclid.trentu.ca/math/sb/misc/mathsci.html. In ultima analisi, tutte le scienze naturali sono fondate sulla matematica. Ad esempio, la sociologia si fonda sulla psicologia, che si fonda sulla biologia, che si fonda sulla chimica, che si fonda sulla fisica, che per ultima si fonda direttamente sulla matematica, che può essere definita la scienza più "pura", perché non studia né gli esseri umani, né la natura materiale, ma fenomeni completamente astratti. V. R. Munroe, *Purity*, n.d., https://xkcd.com/435/.

[72]Intesa come l'applicazione dell'informatica per il trattamento dei dati ai fini processuali.

naturale a pieno titolo.

In primo luogo i supporti materiali su cui i dati sono memorizzati sono soggetti ai fenomeni naturali, e lo studio delle scienze naturali permette di determinare le migliori modalità per il trattamento dei supporti materiali.[73]

Ancora, il sistema operativo ed i programmi possono essere considerati un "fenomeno naturale" perché le loro modalità di funzionamento non sono immediatamente evidenti.

Nel software proprietario non si ha accesso al codice sorgente, ma è possible studiare il suo funzionamento con le stesse tecniche che gli scienziati usano per lo studio dei fenomeni naturali:[74]

- Si interagisce con il software, e si documentano le operazioni svolte, ed i risultati osservati;[75]
- Si formulano delle ipotesi di leggi che descrivono il funzionamento del fenomeno;[76]
- Si sottopongono le ipotesi a verificazione mediante esperimenti, e si documentano i risultati;[77]

[73]Dato che qualsiasi danno al supporto materiale diventa anche un danno per i dati informatici in esso contenuti, l'informatica forense smette di avere una natura puramente astratta, e viene "contaminata" dalle scienze naturali.

[74]Per un esempio di un'opera che applica il metodo scientifico all'informatica forense, v. Mariagrazia Cinti, «Quantificazione ed individuazione delle alterazioni dei dati nell'ambito di indagini di Informatica Forense», *Alma Mater Studiorum – Università di Bologna*, 2011, http://amslaurea.unibo.it/2736/.

[75]È importante documentare in maniera dettagliata non solo le modalità d'uso del software, ma anche le modalità con cui è stato installato e configurato, che versione si sta usando, da dove è stato scaricato, ecc.

[76]Ad esempio, "data la sequenza di azioni X, il programma produce i cambiamenti Y". La formulazione delle ipotesi è libera, e non segue schemi formali, ma è possibile formulare nuove ipotesi iterando su quelle già sviluppate. V. James Blachowicz, «How Science Textbooks Treat Scientific Method: A Philosopher's Perspective», *The British Journal for the Philosophy of Science*, vol. 60, fasc. 2, 2009, https://doi.org/10.1093/bjps/axp011, pp. 303–344, pp. 321–323.

[77]La fase di verificazione è particolarmente delicata. Il solo fatto che i risultati osservati confermano l'ipotesi oggetto di esame non è sufficiente a dimostrare che l'ipotesi sia valida, perché serve anche dimostrare che i risultati non siano dovuti a cause alternative (v. J. Blachowicz, *ivi*, p. 325). È la fallacia logica di affermare il conseguente: "Se A, B; B; pertanto, A", ma questo ignora il fatto che B potrebbe avere altre cause oltre che A. Ad esempio, si potrebbe affermare: "Se nei programmi ci sono bug, si arresteranno in maniera inaspettata. Windows si è arrestato inaspettatamente, pertanto Windows deve avere un bug". Tuttavia, se Windows si arresta anche quando esegue istruzioni estremamente

- Le ipotesi e l'esperimento vengono raffinati, in modo da cercare di creare un esperimento controllato, un esperimento dove l'unico elemento che cambia è la variabile che viene studiata;[78]
- L'esperimento viene condiviso con altri ricercatori, in modo da garantire che sia ripetibile,[79] ed i risultati siano riproducibili.[80] La riproduzione dei risultati degli esperimenti da parte di altri ricercatori rafforza la validità della prima verificazione;[81]
- Al termine del processo, si arriva alla creazione di una serie di massime di esperienza, di "leggi scientifiche" che sono state comprovate empiricamente, e formano una "teoria" sul funzionamento di quel programma.[82]

Nel software libero non è necessario usare queste tecniche, perché è possible studiare il funzionamento del software leggendo il suo codice sorgente. Tuttavia, rimane sempre la possibilità che il codice contenga *bug* (errori di programmazione).[83] I *bug* sono studiati dai programmatori con tecniche di *debugging*, che sono pienamente ispirate al metodo scientifico.[84] L'informatica forense è interessata allo studio dei *bug*

semplici, come impostare un valore a 0, si iniziano a sospettare altre cause per l'arresto inaspettato, tra cui l'instabilità dell'hardware dovuta ad *overclocking* (la sovralimentazione di un processore al fine di aumentare le prestazioni, al costo di sacrificare il suo corretto funzionamento). V. R. Chen, *There's an awful lot of overclocking out there*, 2005, https://web.archive.org/web/20231003201601/https://devblogs.microsoft.com/oldnewthing/20050412-47/?p=35923.

[78]Nell'informatica, si parla di *minimum reproducible example* (minimo esempio riproducibile). L'esperimento deve contenere la minima quantità di azioni strettamente necessarie per raggiungere il risultato, deve contenere gli eventuali dati da fornire in input, e si deve verificare che se eseguito più volte, produca sempre gli stessi risultati. V. Vercel.com, *https://web.archive.org/web/20220927020224/https://vercel.com/guides/creating-a-minimal-reproducible-example*, n.d., https://web.archive.org/web/20220927020224/https://vercel.com/guides/creating-a-minimal-reproducible-example.

[79]Ossia, altre persone possono svolgere le stesse azioni.

[80]Ossia, svolgere le stesse azioni porta agli stessi risultati.

[81]Se un certo comportamento non può essere osservato con regolarità, non può formare la base di teorie scientifiche. A questo fine, è estremamente importante che i ricercatori siano il più trasparenti possibile con la loro ricerca, in modo da permettere ed agevolare un controllo diffuso delle loro teorie.

[82]È importante indicare che "verificare" una teoria non significa che quella teoria è necessariamente corretta, ma solo che riesce a prevedere in maniera affidabile la realtà. La scienza non arriva mai ad "affermare la verità", ma solo a creare modelli che "approssimano la realtà".

[83]Un bug è la situazione che si verifica quando leggendo il codice ci si aspetta il risultato *X*, ma eseguendolo si ottiene il risultato *Y*.

[84]Un bug viene rilevato (osservazione), si documentano le azioni che lo causano (documentazione),

per valutare il loro impatto sui dati.[85]

Un altro elemento da considerare è l'infinita e perfetta riproducibilità dei dati informatici oggetto di analisi. A differenza dei fenomeni naturali, o delle prove materiali, è possibile duplicarli un numero infinito di volte senza *generational loss* (perdita di qualità fra copie successive),[86] ed è sempre possibile verificare l'integrità della copia rispetto all'originale[87] calcolando l'*hash* crittografico dei dati.[88]

Inoltre, dato che i dati informatici devono essere analizzati con programmi informatici, che a loro volta sono composti di dati informatici, anche gli strumenti di analisi sono infinitamente e perfettamente riproducibili. In linea teorica, è sempre possibile ripetere le operazioni di analisi, ottenendo gli stessi risultati.[89]

---

si formula un ipotesi riguardo a quali istruzioni nel codice possano causare quel bug (formulazione di ipotesi), si apportano le modifiche necessarie al codice per vedere se il bug continua a presentarsi (verifica dell'ipotesi), e si continuano a formulare e verificare altre ipotesi fino a quando il bug viene corretto. È buona pratica documentare, dove possibile e ragionevole, la causa del bug, in modo da evitare una *regression* (situazione dove lo stesso bug che era stato già risolto si ripresenta nel futuro), ed evitare di commettere lo stesso errore in futuro in altre parti del codice.

[85]Dal punto di vista dell'informatica forense, un *bug* non è un difetto del programma oggetto di analisi, ma parte integrante del suo funzionamento.

[86]Ad esempio, si pensi a come le fotocopie di fotocopie hanno una qualità minore rispetto ad una fotocopia dell'originale. I dati informatici sono soltanto delle sequenze di valori binari, ed è estremamente semplice creare delle copie, e confrontarle. Se le sequenze di bit sono identiche, i dati informatici sono indistinguibili ed equivalenti, e non è possibile distinguere fra l'originale e la copia.

[87]L'integrità della copia va verificata subito dopo la sua creazione, e periodicamente nel corso del tempo.

[88]In inglese, *to hash* significa "sminuzzare". Un algoritmo di hash "sminuzza" un file, nel senso che il file viene diviso letto come una serie di *blocks* ("blocchi", tranche), che vengono progressivamente ricombinati fra di loro per generare un *digest* (riassunto) dei dati originali, che ha una lunghezza fissa e breve (128 bit per MD5, 160 bit per SHA-1). La prima proprietà degli hash è che gli stessi dati in entrata producono sempre lo stesso hash in uscita. Si può verificare che due sequenze di bit sono identiche calcolando e confrontando il loro hash. La seconda proprietà degli hash è che cambiare anche un singolo bit nei dati in entrata cambierà (in media) la metà dei bit in uscita. Pertanto, anche la minima differenza fra due sequenze di bit produrrà hash completamente diversi. È una buona pratica usare almeno due hash, in modo da avere più valori di riferimento per verificare l'integrità dei dati.

[89]Nella pratica esistono alcune limitazioni. Ad esempio, l'uso degli strumenti di analisi potrebbe essere collegato ad una licenza, e quindi anche se si crea una copia del programma, solo un utente potrebbe utilizzarla. Ancora, mentre è sempre possibile copiare i programmi, potrebbe non essere possibile eseguirli su sistemi operativi o hardware più recente, a causa di incompatibilità. Al di fuori di questi casi-limite, nel caso in cui ripetere la stessa analisi (con gli stessi dati, gli stessi strumenti, lo stesso sistema, la stessa configurazione, ecc.) produca risultati diversi, sarà compito dei tecnici spiegare le differenze, e compito del giudice decidere come valutare i risultati.

1.5 Prova informatica e perizia

Spesso si parla di "prova scientifica" e "prova informatica", ma sono espressioni improprie. Sarebbe più corretto parlare di "prova fondata su teorie scientifiche", e "prova che ha ad oggetto dati informatici".

La perizia è il mezzo di prova ideale per l'introduzione di conoscenze scientifiche all'interno del processo, e dato che l'informatica forense può essere considerata una scienza che studia il trattamento dei dati informatici, si può affermare che la prova informatica tende a coincidere con la perizia. Secondo il codice di procedura penale:[90]

> La perizia è ammessa quando occorre svolgere indagini o acquisire dati o valutazioni che richiedono specifiche competenze tecniche, scientifiche o artistiche.

È preferibile interpretare la disposizione in maniera estensiva, in modo da poter usare la perizia nel maggior numero di casi possibili: il trattamento dei dati informatici può essere sicuramente considerato una situazione che richiede "competenze tecniche" e "scientifiche"[91] e l'espressione "è ammessa" va interpretata come "deve essere ammessa", ed il giudice non può decidere discrezionalmente se ammetterla o meno.[92]

È preferibile evitare di acquisire i dati informatici come se fossero documenti (art. 234 c.p.p.), perché sarebbero inseriti nel fascicolo del dibattimento (art. 515 c.p.p.), ed il giudice ne prenderebbe direttamente cognizione ai fini della decisione (art. 526 co. 1 c.p.p.).[93]

[90] Art. 220 co. 1 c.p.p.

[91] Nei capitoli precedenti si è menzionata la loro fragilità, e la natura scientifica dell'informatica forense. Nei capitoli successivi le esigenze dell'informatica forense, ed in particolare del software utilizzato per il trattamento dei dati, saranno oggetto di discussione più approfondita.

[92] Gli argomenti a favore di questa interpretazione sono il fatto che la perizia possa essere richiesta anche d'ufficio (art. 224 co. 1 c.p.p.), ed l'obbligo di motivazione delle decisioni del giudice (art. 116 co. 6 Cost., art. 546 co. 1 lett. *e* c.p.p.). Se i dati informatici sono rilevanti ai fini della decisione, ma il giudice non è in grado di spiegare in maniera adeguata come sono stati acquisiti e analizzati, la motivazione deve essere ritenuta insufficiente.

[93] In questo caso, esiste il rischio che il giudice valuti i dati così come si presentano visivamente, e non sia in grado di valutare se il documento sia stato acquisito correttamente, se sia stato manipolato, ecc. Questo tipo di ignoranza è scusabile, perché queste conoscenze vanno oltre la conoscenza della persona

È anche preferibile evitare di acquisire i dati informatici come una prova atipica (art. 189 c.p.p.), perché questo tipo di prova è una norma di chiusura del sistema,[94] e può essere utilizzata per le prove scientifiche "nuove" e "controverse",[95] ma è difficile ritenere che l'informatica forense possa rientrare in quelle categorie.[96] Un ulteriore problema della prova atipica è che non sono previste cautele per la corretta acquisizione della prova, garanzie per la tutela dei diritti fondamentali,[97] o sanzioni processuali.[98]

L'espressione "acquisire dati" può essere interpretata nel senso che la perizia possa essere usata anche per acquisire dati informatici, e non solo per valutarli. Pertanto, è preferibile evitare di usare istituti come l'ispezione (art. 244 co. 2 c.p.p.) o perquisizione (art. 247 co. 1-*bis* c.p.p.) di un sistema informatico,[99] o gli accertamenti

---

media, ma è proprio per questo motivo che il giudice è tenuto a richiedere la perizia. Come regola generale, è preferibile partire sempre dal presupposto che la perizia serva, ed escludere il suo utilizzo solo in casi marginali. Ad esempio, il dato informatico viene acquisito come documento, ma viene usato solo *ad abundantiam* per corroborare una ricostruzione dei fatti che è già largamente supportata da prove più affidabili.

[94]Deve essere usata in casi estremi, solo quando le modalità che si intendono usare non sono assolutamente riconducibili ad una prova tipica e già disciplinata dal legislatore. Ad esempio, i dati prodotti dall'operazione di controllo satellitare mediante GPS sono considerati una prova atipica (v. Silvia Renzetti, «La prova scientifica nel processo penale: problemi e prospettive», *Rivista di Diritto Processuale*, vol. 75, fasc. 2, 2015, pp. 399–423, p. 404), perché l'attività di pedinamento in generale è considerata un'attività di indagine atipica (v. Giovanni Conso, Marta Bargis, Vittorio Grevi, *Compendio di procedura penale*, CEDAM, 2020, p. 449).

[95]In questo caso, le parti discutono se la prova scientifica sia effettivamente idonea ad accertare i fatti, in modo da evitare alla radice l'ammissione di *junk science* (pseudoscienza) nel processo (v. S. Renzetti, *op. cit.*, pp. 406–408).

[96]Per quanto l'oggetto della disciplina possa essere in continua evoluzione, è necessario tenere a mente che l'attività di ricerca dell'informatica forense non è nuova o controversa, ma può seguire il modello delle scienze naturali.

[97]L'unico limite è l'inammissibilità di prove atipiche che incidono su libertà fondamentali costituzionalmente garantite (v. G. Conso, M. Bargis, V. Grevi, *op. cit.*, p. 449).

[98]Viceversa, esistono già altri istituiti che offrono garanzie maggiori per la persona e per i dati, ed è molto più opportuno usare quelli.

[99]L'ispezione serve ad "accertare le tracce e gli altri effetti materiali del reato" (art. 244 co. 1 c.p.p.), mentre la perquisizione serve a cercare il "corpo del reato o cose pertinenti al reato" (art. 247 co. 1 c.p.p.). In entrambi i casi, si devono usare "misure tecniche dirette ad assicurare la conservazione dei dati originali e ad impedirne l'alterazione", ed il modo migliore per farlo è acquisire una copia dei dati contenuti nel dispositivo. Tuttavia, dato che la copia di dati informatici può essere considerato un sequestro, perché riguarda l'acquisizione del "corpo del reato", definito come "le cose sulle quali o mediante le quali il reato è stato commesso" (art. 253 c.p.p.), i due istituti tendono a sovrapporsi. Se in entrambi i casi si arriva comunque alla copia dei dati, è meglio eseguirla fin dall'inizio mediante l'intervento di un perito, e svolgere tutte le operazioni successive sulla base di quella copia, invece di procedere a più acquisizioni nel tempo.

tecnici irripetibili.[100]

La "valutazione" dei dati può essere svolta dal solo perito, dai soli consulenti tecnici delle parti[101] o, nel caso ideale, dal perito e dai consulenti.[102]

Il perito è imparziale: i suoi compiti sono di valutare l'attendibilità dei dati[103] e rispondere ai requisiti formulati dal giudice[104] nel modo più completo ed oggettivo possibile.[105]

Viceversa, i consulenti tecnici devono valutare gli stessi elementi a disposizione del perito, ma devono favorire la parte assistita. Se le conclusioni del perito sono favorevoli alla parte, i consulenti tecnici cercheranno altri argomenti a supporto.[106] Se le conclusioni sono sfavorevoli, i consulenti tecnici dovranno cercare di screditare

[100]È possibile qualificare gli accertamenti relativi ai dati informatici come irripetibili perché i dati informatici possono essere "soggett[i] a modificazione" (art. 360 c.p.p.) o perché l'accertamento potrebbe "determina[re] modificazioni delle cose" (art. 127 disp. att. c.p.p.). In linea generale, sarebbe sempre preferibile usare la perizia, perché è la modalità più garantita. Tuttavia, laddove questo sia assolutamente impossibile, ed esiste un rischio concreto che i dati siano alterati o dispersi prima che si possa svolgere la perizia all'interno di un incidente probatorio, è possibile acquisirli mediante accertamento tecnico irripetibile. È prevista un'importante sanzione processuale: se in seguito si dimostra che era possibile attendere l'incidente probatorio, e che quindi l'accertamento è stato fatto con modalità irripetibili senza reale necessità, i risultati dell'accertamento sono inutilizzabili (art. 360 co. 4 e 5 c.p.c).

[101]Le parti possono nominare un proprio consulente tecnico anche se non viene disposta la perizia. In questo caso, i consulenti potranno intervenire alle ispezioni (non si menziona la possibilità di partecipare alle perquisizioni, ma è possibile un'applicazione analogica della norma, dato che andrebbe a favore dell'indagato o imputato), potranno esaminare le cose che sono state oggetto di ispezione o perquisizione, e potranno presentare il loro parere anche con memorie (art. 233 co. 1–1-*ter* c.p.p.).

[102]In questo caso, i consulenti potranno assistere al conferimento dell'incarico e partecipare alle operazioni peritali, possono presentare richieste, osservazioni e richieste (che devono essere verbalizzate), e se vengono nominati dopo lo svolgimento della perizia, possono esaminare l'oggetto della perizia e la relazione del perito (art. 230 c.p.p.). In particolare, se partecipano al conferimento dell'incarico, il giudice formula i quesiti al perito dopo aver sentito anche i consulenti tecnici (art. 226 c.p.p.).

[103]Deve verificare se le modalità di acquisizione hanno acquisito tutti i dati o solo parte di essi, se l'originale è stato modificato nel processo di acquisizione, e se la copia è conforme all'originale; successivamente, deve verificare la presenza di irregolarità nei dati, che potrebbero indicare che i dati siano stati manipolati per ostacolare o sviare le indagini, e indicare le possibili cause di queste irregolarità.

[104]In particolare, deve indicare gli strumenti utilizzati, le procedure seguite, il significato delle operazioni, e l'attendibilità dei risultati.

[105]In particolare, deve indicare tutti i possibili decorsi causali che possono spiegare l'assetto dei dati informatici, sia che siano a favore, sia che siano a sfavore della parte.

[106]Ad esempio, compieranno le stesse analisi con strumenti diversi per dimostrare che si arriva agli stessi risultati, o dimostreranno l'assenza di decorsi causali alternativi che avrebbero potuto portare allo stesso risultato.

l'analisi del perito.[107] L'attività del consulente tecnico del difensore deve essere necessariamente favorevole all'assistito,[108] ma non può spingersi fino alla commissione di un reato.[109]

Il perito, e a maggior ragione i consulenti tecnici, possono anche arrivare a "conclusioni divergenti rispetto all'opinione comune corrente"[110] purché motivino adeguatamente il loro operato e le loro conclusioni durante il contraddittorio.[111]

Il giudice è *peritus peritorum* (perito dei periti), e quindi è libero di valutare le contribuzioni del perito e dei consulenti tecnici. È importante che il giudice eviti di cadere in due situazioni estreme: una dove non richiede la prova scientifica anche quando sarebbe opportuno, e una dove rimette la decisione interamente agli esperti.[112] Il giudice è tenuto a valutare tutti i contributi in maniera libera e critica, senza

[107] Ad esempio, i dati sono stati acquisiti usando tecniche inadeguate e quindi l'analisi è viziata *ab origine*; i risultati sono inaffidabili perché il perito ha usato modalità di analisi originali, nuove e non sufficientemente condivise dalla comunità scientifica, inadeguate al caso concreto, che non rispettano i principi fondamentali dell'informatica forense, ecc.; è possibile ottenere un risultato diverso e più affidabile usando metodi di analisi, strumenti o procedure diverse; esistono ulteriori fattori che il perito non ha considerato, o a cui non ha dato sufficiente peso, che possono spiegare lo stato dei dati informatici. In generale, è sufficiente che il consulente tecnico del difensore generi un "ragionevole dubbio" riguardo la prova del fatto per evitare una sentenza di condanna (art. 533 co. 1 c.p.p.), e non è necessario provare un fatto nuovo e diverso.

[108] Se causasse un danno con dolo, integrerebbe il reato di consulenza infedele (art. 380 c.p.) (v. Stefano Canestrari et al., *Diritto penale. Lineamenti di parte speciale*, Monduzzi Editoriale, 2016, p. 273).

[109] Ad esempio, le attività materiali che ostacolano le indagini (come distruggere o manipolare le prove) integrerebbero il reato di favoreggiamento personale (art. 378 c.p.) (v. S. Canestrari et al., *ivi*, p. 252).

[110] *Ibidem*, p. 265.

[111] Si potrebbe affermare che la funzione del perito e dei consulenti è di garantire la più piena attuazione del diritto di difesa, ed in particolare, dell'art. 111 co. 4 Cost., secondo cui nel processo penale la prova si forma in contraddittorio. Per quanto riguarda le prove scientifiche (in questo caso, la prova che ha per oggetto dati informatici) è importante garantire che la prova si formi all'interno di un contraddittorio tecnico. Nel caso ideale, entrambe le parti si doteranno di consulenti prima della perizia, e il perito sarà il primo soggetto che andrà a interagire con i dati.

[112] Nel primo caso, il giudice può decidere discrezionalmente se serva l'intervento di un perito, o se siano sufficienti le conoscenze della persona media. Tuttavia, come discusso, è generalmente preferibile l'intervento di un perito quando si usano dati informatici. Il secondo caso è particolarmente insidioso se nessuna delle parti, o solo una di loro (presumibilmente il PM, dato che la parte pubblica ha risorse maggiori) ha nominato un consulente tecnico. V. S. Renzetti, *op. cit.*, pp. 405–406.

creare gerarchie fra i tecnici[113] e fra i tipi di prova.[114] È importante che il giudice valuti la ragionevolezza intrinseca dell'attività svolta, e la sua coerenza con i principi fondamentali della materia, specie se è stato necessario usare tecniche di analisi complesse o innovative.[115]

[113]Ad esempio, non deve preferire le conclusioni del perito rispetto a quelle dei consulenti tecnici, e non deve preferire il consulente del PM rispetto al consulente della difesa, argomentando che sono "più oggettivi", altrimenti si annichilisce il senso del modello accusatorio, e si ritorna al modello inquisitorio.

[114]Anche qui il giudice deve evitare due estremi, uno dove la prova informatica prevale sulle altre prove solo perché è una prova scientifica (v. S. Renzetti, *op. cit.*, p. 412), e uno dove la prova informatica non viene accolta, solo perché è stato necessario usare una tecnica o strumento di analisi originale, che non è stato oggetto di larga discussione (v. Giorgio Marinucci, Emilio Dolcini, Gian Luigi Gatta, *Manuale di Diritto Penale. Parte Generale. Nona edizione*, Giuffrè Francis Lefebvre, 2020, p. 246). Il secondo caso si può verificare perché l'informatica è una materia complessa ed in continua evoluzione, ed nei settori più in avanguardia è possibile che non esistano ancora delle prassi affermate.

[115]Ad esempio, un conto è la ricerca di file per *hash*, che è concettualmente semplice da spiegare (tutti i file all'interno di un supporto vengono confrontati con una lista di file già conosciuti), e produce risultati certi (o il file rientra in quella lista, o non vi rientra; v. M. Ferrazzano, *op. cit.*, p. 153), un conto sono le tecniche di analisi per verificare se un'immagine è stata manipolata, che sono concettualmente complesse, e c'è un margine di valutazione discrezionale (v. Sebastiano Battiato, Giuseppe Messina, Rosetta Rizzo, «Image forensics. Contraffazione digitale e identificazione della camera di acquisizione: status e prospettive», 2014, https://www.researchgate.net/publication/242495487, pp. 16–20 e M. Ferrazzano, *op. cit.*, pp. 152–153).

Capitolo 2

Software libero come modello ideale per l'informatica forense

2.1 Esigenze del software per l'informatica forense

2.1.1 Acquisizione dei dati informatici

Il trattamento dei dati informatici all'interno dell'informatica forense è diviso in una serie di fasi. Le prime due fasi hanno natura puramente materiale, e non richiedono l'uso di software specializzato: sono l'identificazione[1] e la raccolta.[2]

L'uso di software specializzato, che deve essere conforme alle indicazioni elaborate dall'informatica forense, diventa necessario a partire dall'acquisizione, la fase in cui si crea una copia dei dati informatici.[3]

Quando il codice di procedura penale menziona operazioni che potrebbero influire

[1] È la fase in cui si ricercano i supporti materiali che possono contenere dati informatici utili, v. M. Ferrazzano, *op. cit.*, pp. 29–30.

[2] È la fase in cui i supporti materiali, e qualsiasi altro oggetto necessario per il loro funzionamento, o comune utile per le indagini, viene rimosso fisicamente. In alcuni casi non è possibile procedere alla raccolta (ad esempio, nel caso di sistemi informatici che erogano servizi essenziali, e quindi devono rimanere sempre accesi), e pertanto si deve passare immediatamente alla fase di acquisizione. V. M. Ferrazzano, *ivi*, pp. 30–34.

[3] *Ibidem*, p. 34.

sull'integrità dei dati informatici[4] usa espressioni come "adottando misure tecniche dirette ad assicurare la conservazione dei dati originali e ad impedirne l'alterazione"[5] oppure "mediante procedura che assicuri la conformità della copia all'originale e la sua immodificabilità".[6]

La procedura di acquisizione dei dati informatici deve impedire l'alterazione dei dati originali, deve garantire che la copia sia conforme all'originale e deve permettere di rilevare se la copia sia stata modificata.[7]

I dati possono essere acquisiti da fonti diverse[8] e ogni fonte presenta caratteristiche diverse, che influenzano come i dati devono essere acquisiti.

Il primo passo è valutare la quantità e qualità dei dati che è possibile acquisire dal supporto. Nel caso di supporti materiali estraibili[9] è generalmente possibile acquisire tutti i dati che contengono.[10] Nel caso di supporti materiali *embedded* (integrati, e quindi non estraibili)[11] e nel caso della *network forensics*[12] è possibile acquisire meno dati.[13] Se un sistema informatico è acceso, è possibile acquisire i contenuti della sua

[4]Come le ispezioni, perquisizioni, sequestri e accertamenti urgenti.

[5]Art. 244 co. 2, art. 247 co. 1-*bis* e art. 352 co. 1-*bis* c.p.p.

[6]Art. 254-*bis*, art. 354 co. 2 c.p.p.

[7]È praticamente impossible garantire la vera immodificabilità dei dati informatici, il meglio che si può fare è usare strumenti che verificano se sono stati modificati o meno.

[8]Le fonti includono i supporti materiali rimovibili, i supporti integrati, i siti e servizi accessibili online, la RAM, ecc.

[9]Ad esempio, dischi rigidi o memorie *flash* (chiavette USB, schede SD, e SSD), ecc.

[10]È importante che il software per l'informatica forense implementi correttamente le specifiche tecniche relative al funzionamento di ogni tipo di supporto materiale, in modo che sia possibile usare i comandi di basso livello (ossia, che permettono di interagire direttamente con l'hardware) per garantire l'estrazione della maggiore quantità di informazioni possibili (ad esempio, informazioni diagnostiche). In ogni caso, è importante catturare anche lo spazio non allocato (ossia, non assegnato ad una partizione), e lo slack space (lo spazio libero all'interno delle singole partizioni), perché potrebbero contenere tracce di dati informatici. V. M. Ferrazzano, *op. cit.*, p. 34.

[11]Il caso tipico sono i supporti di memoria contenuti negli smartphone, tablet e nei computer più recenti della Apple (non sono rimovibili perché saldati alla scheda madre del dispositivo), oppure dei dispositivi *ad hoc* che non usano parti standard (ad esempio, apparecchiature mediche).

[12]Analisi forense di dati che sono resi disponibili mediante connessioni di rete. Ad esempio, siti internet, dati conservati su piattaforme cloud, dati conservati su altri sistemi informatici a cui non si ha un accesso diretto, ed è necessario usare protocolli di rete (come HTTP(S), BitTorrent, SMB, SSH, ecc.) per accedervi, ecc.

[13]In entrambi i casi l'unica modalità per acquisire dati informatici è di comunicare con il software che viene eseguito sul dispositivo integrato o server remoto. Il problema con questo tipo di acquisizioni è che il software con cui si interagisce è libero di nascondere o modificare i dati informatici. È generalmente sconsigliabile cercare di eseguire operazioni di forzatura delle misure di protezione per cercare di ottenere

memoria volatile.[14]

Il secondo passo è impedire che i dati originali vengano modificati durante l'operazione di copia. Pertanto, il software per l'informatica forense deve usare tutte le precauzioni possibili per ridurre al minimo questo rischio.[15]

Il terzo passo è creare una *forensic image* (copia forense) dei dati.[16] Il problema è la corretta gestione degli errori di lettura dei dati da parte del software per l'informatica forense.[17]

L'ultimo passo è verificare la conformità della copia all'originale. Questa operazione

---

il pieno accesso a tutti i dati contenuti sul dispositivo, perché si va ad alterare il normale funzionamento del software, e quindi si potrebbero danneggiare o modificare i dati in maniera difficilmente prevedibile. Piuttosto, è preferibile accontentarsi di leggere i dati che il dispositivo mette naturalmente a disposizione, tenendo sempre a mente che le informazioni offerte potrebbero non essere complete o precise. Ad esempio, il protocollo HTTP (v. R. Fielding, M. Nottingham, J. Reschke, *RFC 9110: HTTP Semantics*, 2022, https://httpwg.org/specs/rfc9110.html) è largamente usato per trasferire singoli file. Tuttavia, il protocollo non richiede la comunicazione corretta degli elementi più basilari del file. Ad esempio, l'ultima modifica del file su disco non deve essere riprodotta nel campo *Last-Modified*, ed l'indicazione del nome del file quando viene scaricato è un'estensione allo standard base (v. J. Reschke, *RFC 6266: Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)*, 2011, https://httpwg.org/specs/rfc6266.html).

[14] È importante catturare l'intero contenuto della memoria. Quando la RAM si riempie, il sistema operativo trasferisce parte dei dati della memoria all'interno di un file o partizione di *swap* (nei sistemi Linux) o nel *pagefile* (nei sistemi Windows). Pertanto, è importante acquisire anche una copia di questi file. I contenuti della RAM includono i programmi in esecuzione, gli utenti connessi, le connessioni di rete attive, i dispositivi in uso, ecc. V. M. Ferrazzano, *op. cit.*, p. 36.

[15] Ad esempio, quando si apre un singolo file è preferile usare la funzione *open*, perché permette di specificare in maniera esplicita le opzioni *O_RDONLY* (apertura in sola lettura) e *O_NOATIME* (divieto di cambiare la data di ultima apertura del file). Queste opzioni impediscono la modifica dei dati o metadati del file (v. M. Kerrisk, *open(2) — Linux manual page*, 2023, https://web.archive.org/web/20231209031506/https://man7.org/linux/man-pages/man2/open.2.html, sez. "DESCRIPTION"). Quando si collega un supporto materiale è importante usare prima il comando *blockdev* con l'opzione *setro* per bloccare qualsiasi operazione in scrittura (v. Z. Karel, *mount(8) Manual Page*, 2023, https://github.com/util-linux/util-linux/blob/6b081fa421f4028bad0be22178f8d8e5e9015041/sys-utils/mount.8.adoc, sez. "COMMAND-LINE OPTIONS") e poi il comando *mount* con le opzioni *ro* (sola lettura), *noatime* (divieto di aggiornare la data di ultima apertura), e *noexec* (divieto di eseguire file) (v. Eva Huebner, Stefano Zanero, *Open Source Software for Digital Forensics*, Springer Science+Business Media, 2010, p. 73).

[16] Si parla anche di *bitstream image* (copia bit-a-bit), ma l'espressione non va intesa letteralmente: sarebbe estremamente inefficiente copiare un bit alla volta. La copia viene eseguita leggendo e copiando blocchi di centinaia o migliaia di byte alla volta, dove un byte sono 8 bit.

[17] Il software deve essere in grado di rilevare che sia avvenuto un errore, e deve chiedere all'utente come proseguire (se riprovare ad acquisire il dato, se saltare l'acquisizione del dato che ha causato l'errore e continuare l'operazione, o se interrompere l'operazione). Il programma deve fornire quante più informazioni utili, in modo che l'utente possa prendere una decisione informata. Il programma deve tenere traccia delle operazioni svolte, delle decisioni dell'utente, e di qualsiasi evento che possa influire sulla quantità o qualità dei dati.

si svolge confrontando l'hash dei dati originali con l'hash della copia appena creata[18] usando almeno due funzioni di hash.[19]

La verifica è possibile solo se l'operazione di acquisizione è ripetibile, perché richiede una seconda lettura dei dati informatici già acquisiti.[20] La fase di verificazione della copia non pone difficoltà significative.[21] Il calcolo dell'hash della copia può essere anche ripetuto a distanza di tempo, per verificare se la copia abbia subito modificazioni.[22]

2.1.2 Conservazione dei dati informatici

La conservazione[23] è la fase in cui i supporti materiali vengono preparati per il trasporto e i dati vengono preparati per l'archiviazione di lungo termine.[24]

Il codice di procedura fa riferimento alla custodia dei dati informatici in due articoli. L'art. 259 co. 2 c.p.p. recita:

[18]Le funzioni di hash permettono di trasformare una sequenza di dati informatici di lunghezza arbitraria in una sequenza di dati di lunghezza fissa (chiamata *digest*). La proprietà fondamentale è che allo stesso input corrisponde lo stesso *digest*, e pertanto è possibile usare le funzioni di hash per verificare se due file sono identici. V. M. Ferrazzano, *op. cit.*, p. 35.

[19]Esistono metodi per generare lo stesso *digest* (e quindi dare l'illusione che i dati siano uguali) anche se i valori in input sono diversi (è un attacco crittografico chiamato *chosen-prefix collision*, v. Gaëtan Leurent, Thomas Peyrin, «SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust», 2020, https://www.usenix.org/system/files/sec20-leurent.pdf, p. 1839), ma è improbabile che lo stesso metodo funzioni per due funzioni di hash diverse. Il secondo è che nel tempo le funzioni di hash diventano obsolete (v. National Institute of Standards and Technology, *NIST Transitioning Away from SHA-1 for All Applications*, 2022, https://web.archive.org/web/20221216212335/https://csrc.nist.gov/news/2022/nist-transitioning-away-from-sha-1-for-all-apps) ed è usare le funzioni di hash più recenti rende la propria applicazione a prova di futuro.

[20]L'operazione di acquisizione può essere irripetibile per sua natura (ad esempio, i contenuti della RAM cambiano in continuazione) o per le circostanze del caso concreto (ad esempio, un supporto materiale che produce numerosi errori di lettura). In questi casi la è necessario documentare l'operazione anche con strumenti esterni rispetto al software (v. M. Ferrazzano, *op. cit.*, pp. 35–36).

[21]Le specificazioni tecniche relative alle funzioni di hash sono dettagliate (v. National Institute of Standards and Technology, «Secure Hash Standard (SHS)», 2015, http://dx.doi.org/10.6028/NIST.FIPS.180-4), ed esistono esempi ufficiali per verificare la correttezza dell'implementazione (v. National Institute of Standards and Technology, *Examples with Intermediate Values*, 2023, https://web.archive.org/web/20230603170119/https://csrc.nist.gov/Projects/cryptographic-standards-and-guidelines/example-values.

[22]Basta ricalcolare l'hash della copia, e verificare se è cambiato rispetto ai valori che erano stati ottenuti durante la prima verificazione.

[23]M. Ferrazzano, *op. cit.*, p. 37.

[24]È importante notare che la conservazione è una fase particolare, che non si conclude con l'inizio della fase successiva, ma rimane rilevante fino alla conclusione definitiva del trattamento dei dati.

> Quando la custodia riguarda dati, informazioni o programmi informatici, il custode è altresì avvertito dell'obbligo di impedirne l'alterazione o l'accesso da parte di terzi, salva, in quest'ultimo caso, diversa disposizione dell'autorità giudiziaria.

L'art. 260 co. 2 c.p.p. recita:

> L'autorità giudiziaria fa estrarre copia dei documenti e fa eseguire fotografie o altre riproduzioni delle cose sequestrate che possono alterarsi o che sono di difficile custodia, le unisce agli atti e fa custodire in cancelleria o segreteria gli originali dei documenti, disponendo, quanto alle cose, in conformità dell'articolo 259. Quando si tratta di dati, di informazioni o di programmi informatici, la copia deve essere realizzata su adeguati supporti, mediante procedura che assicuri la conformità della copia all'originale e la sua immodificabilità; in tali casi, la custodia degli originali può essere disposta anche in luoghi diversi dalla cancelleria o dalla segreteria.

Per prevenire l'alterazione e l'accesso non autorizzato da parte di terzi, la soluzione più semplice ed efficace è di conservare i supporti materiali all'interno di sistemi informatici che sono *air-gapped*.[25]

L'immodificabilità dei dati può essere garantita con l'uso di software specializzato per il backup e l'archiviazione dei dati, che deve avere una serie di funzionalità come la possibilità di impedire la modifica intenzionale dei dati,[26] di comprimere i dati e proteggerli con crittografia,[27] ed eventualmente di cercare di riparare i dati in caso di

[25] L'*air-gapping* (vuoto d'aria) consiste nello scollegare un sistema informatico dalla rete. Il sistema rimane acceso, e può monitorare la condizione dei dati in tempo reale, ma è impossibile modificare o danneggiare i dati con attacchi informatici.

[26] Ad esempio, permettere solo la lettura dei dati (modalità *read-only*).

[27] La compressione permette di archiviare i dati usando meno spazio. La tecniche di *authenticated encryption* (crittografia autenticata) permettono di cifrare i dati e garantire simultaneamente che non siano stati alterati, perché la decifrazione fallirebbe con un errore se i dati criptati venissero alterati per qualsiasi motivo.

corruzione.[28]

2.1.3 Catena di custodia

La catena di custodia è un documento che inizia ad essere redatto dopo il sequestro dei dati digitali, e continua ad essere aggiornato durante tutta la fase della loro conservazione. La redazione di una catena di custodia è richiesta ed è regolata dagli standard ISO,[29] ma non è menzionata nel codice di procedura italiano.[30]

È possibile prendere come esempio il codice di procedura penale colombiano per esaminare i requisiti processuali della catena di custodia.[31]

L'art. 254 indica la finalità e gli elementi che devono essere considerati:

> *Con el fin de demostrar la autenticidad de los elementos materiales probatorios y evidencia física, la cadena de custodia se aplicará teniendo en cuenta los siguientes factores: identidad, estado original, condiciones de recolección, preservación, embalaje y envío; lugares y fechas de permanencia y los cambios que cada custodio haya realizado. Igualmente se registrará el nombre y la identificación de todas las personas que hayan estado en contacto con esos elementos.*[32]

Per quanto riguarda la prova informatica, è importante redigere la catena di

[28] Ad esempio, duplicando più volte i dati ed usando un *consensus algorithm*, oppure usando un sistema di *error-correcting codes*, ecc.

[29] In particolare, lo standard ISO 27037:2012. V. M. Ferrazzano, *op. cit.*, p. 38

[30] Gli standard ISO non sono vincolanti dal punto di vista legale, e al più possono rilevare come strumenti di *soft law*, che possono ispirare il legislatore ad adottare buone pratiche. Le uniche formalità previste dal codice a seguito del sequestro sono l'applicazione e rimozione di sigilli da parte dell'ufficiale giudiziario sulle cose sequestrate (artt. 260 e 261 c.p.p.).

[31] La disciplina della catena di custodia è contenuta nell'art. 254 e ss. V. Congreso de la República de Colombia, *Ley 906 de 2004, Por la cual se expide el Código de Procedimiento Penal. (Corregida de conformidad con el Decreto 2770 de 2004)*, 2004, https://web.archive.org/web/20200707152305/https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=14787.

[32] Al fine di dimostrare l'autenticità degli elementi di prova e delle prove materiali, la catena di custodia sarà redatta tendendo conto dei seguenti elementi: identità, stato originale, condizioni al momento della raccolta, conservazione, imballaggio, spedizione; luoghi e date di permanenza, e cambiamenti apportati da ogni custode. Allo stesso modo si registrerà il nome ed i documenti di tutte le persone che sono state in contatto con questi elementi.

custodia considerando la doppia natura dei dati informatici, materiale[33] e digitale.[34]

La redazione della catena di custodia è di responsabilità dei funzionari pubblici (art. 255):

> *La aplicación de la cadena de custodia es responsabilidad de los servidores públicos que entren en contacto con los elementos materiales probatorios y evidencia física.*[35]

Per identificare i funzionari pubblici, i custodi, e più in generale, tutte le persone che entrano in contatto con i dati informatici, è possibile redigere una catena di custodia in formato digitale e applicare firme digitali.[36]

Il materiale raccolto deve essere conservato e consegnato il prima possibile al perito. I vari funzionari che man mano entrano in possesso degli elementi sono responsabili per la loro conservazione e devono verificare che i dati non siano stati alterati.[37] Il perito controlla l'integrità dei dati prima di analizzarli (artt. 257 a 261). Dopo l'analisi, si può disporre la distruzione degli elementi o la loro conservazione (art. 262):

> *Los remanentes del elemento material analizado, serán guardados en el almacén que en el laboratorio está destinado para ese fin. Al almacenarlo será*

[33]I supporti materiali sono identificati con il loro numero seriale. Le loro condizioni al momento della raccolta (ad esempio, "inserito in un computer trovato acceso", oppure "disco rigido esterno, non collegato ad un computer") servono per determinare le ultime cause di modificazione che possono aver influenzato i dati. Per quanto riguarda le modalità di acquisizione, conservazione e trasporto, si indica come il supporto è stato estratto, se si sono verificati problemi durante l'estrazione, e quali misure sono state prese per proteggerlo da danni.

[34]I dati informatici sono identificati dai loro hash. Le condizioni al momento della raccolta sono indicate nel log della procedura di acquisizione (che indicherà informazioni come quale programma è stato usare per compiere l'acquisizione, quando l'operazione è stata compiuta, su quale supporto, quanti dati sono stati copiati, se sono avvenuti errori durante la lettura o verificazione, i vari *digests*, ecc.). Per quanto riguarda la loro conservazione, si devono indicare quale precauzioni e quali programmi sono stati usati per garantire l'integrità e cercare di evitare la modifica dei dati.

[35]I funzionari pubblici che entrano in contatto con gli elementi di prova e le prove materiali sono responsabili per la redazione della catena di custodia.

[36]Se la redazione della catena di custodia viene interpretata come un'attività che attribuisce la qualifica di PU, si tratterà di firma digitale autenticata (art. 25 CAD). I soggetti diversi dal PU possono apporre la firma in maniera digitale, o autografa (nel secondo caso si firmerà digitalmente la scansione della catena di custodia cartacea).

[37]Si deve verificare che i supporti materiali siano stati imballati correttamente, e che il supporto materiale ed gli hash dei dati in esso contenuto corrispondano a quanto riportato all'inizio della catena di custodia.

previamente identificado de tal forma que, en cualquier otro momento, pueda ser recuperado para nuevas investigaciones o análisis o para su destrucción, cuando así lo disponga la autoridad judicial competente.[38]

La soluzione ideale per la conservazione dei dati a lungo termine è di cifrarli, e di conservare la chiave di cifratura su carta, in un luogo separato e diverso.[39]

Infine, si prevede che ad ogni passaggio i responsabili si identifichino e diano un resoconto dello stato degli elementi (artt. 263 e 264), e che la catena di custodia sia autenticata dalla polizia giudiziaria e dai periti (art. 265).[40]

Il codice non definisce in cosa consista l'autenticazione. Se la catena di custodia viene redatta in forma digitale si può ragionevolmente pensare che sia necessario verificare l'integrità e completezza della catena, e la validità delle firme digitali.

In generale, si può osservare che gli articoli del codice sono ispirati ad una netta divisione delle responsabilità fra i vari soggetti,[41] che va ad enfatizzare il ruolo del perito e l'importanza della conservazione della prova.[42]

2.1.4 Analisi e valutazione dei dati informatici

L'analisi e la valutazione sono due fasi distinte, ma strettamente legate fra di loro. L'analisi consiste nella ricerca degli elementi che risultano utili per la ricostruzione

[38] I residui degli elementi materiali analizzati saranno conservati nel magazzino che è destinato a questo scopo nel laboratorio. Al momento del deposito, si provvederà ad identificare i residui in modo che, in qualsiasi altro momento, sia possibile recuperarli per nuove investigazioni o analisi o per distruggerli quando richiesto dall'autorità giudiziale competente.

[39] La distruzione della chiave è funzionalmente equivalente alla distruzione dei dati, perché li rende illeggibili.

[40] Per quanto riguarda l'identificazione dei responsabili, v. nota 36. Ogni aggiunta alla catena di custodia deve essere a sua volta firmata dai soggetti presenti.

[41] La polizia giudiziaria ha il solo compito di assicurare la fonte di prova, i soggetti intermedi fra polizia giudiziaria e perito hanno il compito di assicurare la sua corretta conservazione e trasporto, ed il perito è l'unico soggetto che può svolgere accertamenti ed analisi.

[42] La prova deve essere portata al perito nel minor tempo possibile, e tutti i custodi intermedi devono garantire che la prova sia ancora integra. Questa impostazione è particolarmente adatta ai dati digitali, dato che possono essere modificati con facilità ad ogni passaggio di mani, e rilevare queste modifiche non è facile.

del fatto,[43] mentre la valutazione consiste nella interpretazione di questi elementi.[44] Entrambe le fasi sono svolte sia dal perito,[45] sia dai consulenti tecnici.[46]

La valutazione (come fase del trattamento dei dati informatici, svolta da soggetti con competenze tecniche) non va confusa con la libera valutazione delle prove, svolta dal giudice (art. 192 co. 1 c.p.p.), perché hanno natura diversa.[47]

Queste fasi comportano una serie di esigenze per il software. La più importante è che lo strumento di analisi si conformi ai risultati prodotti dalla ricerca scientifica.[48] Laddove questo non sia possibile[49] è necessario argomentare la ragionevolezza

[43]Nel caso più semplice, questi elementi possono essere individuati ed aperti senza l'uso di tecniche particolari; se i file sono stati cancellati, è possibile esaminare l'intero contenuto del disco per cercare di recuperarli; se i file sono stati occultati mediante tecniche di stenografia (nascondere un file dentro un altro file) o resi illeggibili mediante la crittografia, è possibile usare strumenti di analisi specializzati per rilevare l'uso di queste tecniche; se la macchina è stata colpita da un attacco informatico, è necessario verificare la presenza del malware, o di sue tracce, ecc. V. M. Ferrazzano, *op. cit.*, pp. 39–40.

[44]È possibile dare più interpretazioni degli stessi dati informatici (ad esempio, "certi file sono stati scaricati intenzionalmente", oppure "quei file non sono mai stati scaricati, ma sono stati aggiunti da un terzo"). Per ogni interpretazione, si devono cercare gli indizi rilevanti, valutare la loro affidabilità individuale, ed infine, valutare l'affidabilità complessiva dell'interpretazione. Ad esempio, si può verificare se esiste traccia di quei file nella cronologia o cache del browser; se non si trova nulla, si cerca di determinare se la cronologia o cache sono state manipolate per eliminare le tracce; si confrontano i metadati dei file, come la data di creazione o l'utente che ha creato il file, con altri file vicini, alla ricerca di incongruenze; si controlla la data di ultima apertura del file; si ricercano elementi che possono aiutare a determinare se quelle date sono state manipolate; ecc. V. M. Ferrazzano, *ivi*, p. 41.

[45]L'analisi del perito è limitata agli elementi necessari per rispondere ai quesiti che sono stati posti dal giudice, e la sua valutazione deve essere terza ed imparziale (non deve né accusare, né difendere l'imputato).

[46]L'analisi e le valutazioni del consulente tecnico del PM riguardano principalmente gli elementi a sostegno dell'accusa, ma non possono ignorare gli elementi a favore dell'indagato o imputato (cfr. art. 358 c.p.p.). L'analisi e le valutazioni del consulente tecnico del difensore riguardano esclusivamente gli elementi a favore dell'indagato o imputato.

[47]I tecnici valutano direttamente ed esclusivamente i dati informatici, mentre il giudice valuta la loro interpretazione, considerando anche il resto del quadro probatorio. In particolare, è importante ricordare che il perito può solo aiutare il giudice a decidere, ma non può decidere al posto del giudice. Il giudice non può mai compiere un rinvio generico alle valutazioni del perito, oppure semplicemente ripeterle parola per parola. Piuttosto, deve sempre spiegare con proprie parole perché è d'accordo con il perito, e soprattutto, spiegare i motivi per cui non ha accolto le critiche dei consulenti tecnici. Allo stesso modo, il perito deve sempre presentare le sue conclusioni in maniera critica, evidenziando tutte le ragioni per cui una certa interpretazione potrebbe essere inaffidabile. Il ruolo dei consulenti tecnici è svolgere ulteriori critiche e osservazioni nei confronti dell'opera del perito, e di fornire più elementi di valutazione al giudice.

[48]La ricerca scientifica presenta il vantaggio di essere stata discussa e comprovata in maniera empirica e pubblica, mediante un processo imparziale di *peer-review*.

[49]Ad esempio, perché i metodi di analisi sono innovativi o sperimentali (e quindi la comunità scientifica non è ancora arrivata ad un grado soddisfacente di *peer-review*), perché sono proprietari (e

dell'approccio usato, caso per caso.[50]

La seconda esigenza è il corretto funzionamento dello strumento di analisi.[51] Idealmente si dovrebbe ridurre la quantità di bug[52] al minimo, ma è più realistico[53] cercare di fornire la maggiore quantità di documentazione relativa ai bug possibile.[54]

Infine, l'ultima esigenza è la flessibilità dello strumento di analisi. È preferibile che ogni strumento si specializzi per l'analisi di un certo tipo di dati,[55] e che all'interno di quell'ambito offra una varietà di metodi di analisi,[56] anche mediante l'uso di *plug-in* (moduli aggiuntivi) scritti da sviluppatori di terze parti.[57]

---

quindi non si vuole rendere pubblico il loro funzionamento nei dettagli), oppure perché sono basati su tecnologie che per loro natura sono opache (ad esempio, le intelligenze artificiali che sono state addestrate a classificare il contenuto delle immagini).

[50]In generale, si deve sempre spiegare il funzionamento del software, e fornire l'accesso ai dati e alle risorse che sono stati usati per la sua creazione. Più questi elementi sono irragionevoli (ad esempio, si discostano dai principi fondamentali della materia, senza indicare il motivo), generici (ad esempio, gli sviluppatori di un approccio proprietario si rifiutano di entrare nei dettagli per mantenere il segreto industriale), o incompleti (ad esempio, potrebbe essere legalmente o moralmente impossibile fornire le immagini che sono state usate per addestrare un'intelligenza artificiale a riconoscere foto e video pedo-pornografici), e più il giudice deve valutare questi approcci con sfavore.

[51]Dopo che si è dimostrato che il metodo di analisi ha fondamenti scientifici, o che comunque l'approccio seguito è ragionevole, si deve dimostrare che il metodo di analisi è stato implementato correttamente nel software.

[52]Errori di programmazione che portano il programma a compiere operazioni inaspettate o indesiderate, e quindi compromettono la correttezza dei risultati dell'analisi, e più in generale, il suo grado di attendibilità.

[53]Individuare, riprodurre e correggere i bug è un'operazione complessa, che richiede una grande quantità di tempo e risorse, e spesso si parla di *known bugs* (bug conosciuti) per indicare il fatto che i programmatori sono a conoscenza dell'esistenza di un bug, ma non lo hanno corretto subito, per una varietà di motivi (hanno preferito concentrarsi su altri bug più gravi, non sono ancora certi su come correggere il bug, correggerlo richiederebbe un investimento di risorse notevole, ecc.).

[54]Si devono indicare informazioni come quali versioni del software sono viziate dal bug, in che situazioni si verifica, che effetti produce, ecc. Questa documentazione deve rimanere disponibile anche dopo che il bug viene corretto in una versione successiva del software, nel caso in cui si debba riesaminare o ripetere un'analisi svolta con una versione precedente.

[55]Invece che disperdere e duplicare gli sforzi fra numerosi programmi generici, è preferibile concentrarli su un numero ristretto di programmi altamente specializzati.

[56]Il primo motivo è che è difficile prevedere in anticipo i quesiti che potrebbero essere posti dal giudice all'interno di un caso concreto, e pertanto è meglio avere quanti più strumenti possibile a disposizione. Il secondo motivo è che se più metodi di analisi hanno la stessa funzione, e operano secondo tecniche diverse, e queste tecniche sono tutte valide, è possibile confrontare i vari risultati per giungere ad una valutazione più ponderata. Ad esempio, *VirusTotal* permette di analizzare un file usando più di 70 programmi antivirus. Se un ristretto numero di programmi ritiene che il file sia un virus, ma gli altri non rilevano nulla, molto probabilmente si tratta di un falso positivo. V. VirusTotal, *How it works*, 2023, https://web.archive.org/web/20231231202321/https://docs.virustotal.com/docs/how-it-works.

[57]Gli sviluppatori di terze parti sono gli sviluppatori diversi dagli sviluppatori originali. Se gli

2.1.5 Presentazione delle conclusioni e contraddittorio

La presentazione è la fase finale del trattamento dei dati informatici, in cui le valutazioni svolte dal personale tecnico vengono concretamente acquisite all'interno del dibattimento.[58] Questa fase non richiede l'uso di software per il trattamento dei dati, perché è puramente incentrata sulla discussione di come il software è stato usato nelle fasi precedenti.[59]

Le linee-guida per l'ingresso di conoscenze scientifiche all'interno del processo sono state indicate per la prima volta nel 1923 in *Frye v. United States*,[60] e sono state riformulate nel 1993 dalla Corte Suprema degli Stati Uniti in *Daubert v. Merrel Dow Pharmaceuticals*.[61]

In *Daubert* si afferma che il giudice è tenuto a compiere due valutazioni. La prima riguarda l'ammissibilità della prova scientifica,[62] che deve essere fondata su conoscenze

---

sviluppatori originali non sviluppano più il programma in maniera attiva, gli autori di *plug-in* possono continuare a fornire strumenti di analisi migliori e aggiornati, e quindi mantenere in vita il software, quasi sostituendosi agli sviluppatori originali.

[58] V. M. Ferrazzano, *op. cit.*, pp. 41–42. Il perito e gli eventuali consulenti tecnici vengono inseriti nelle liste testimoniali (art. 468 c.p.p.), ed in ogni caso, il giudice acquisisce la relazione finale del perito (art. 227 c.p.p.) e le memorie scritte dai consulenti tecnici (art. 233 co. 1 c.p.p.).

[59] Il perito ed i consulenti discutono gli strumenti di analisi, i loro fondamenti scientifici, e la validità ed affidabilità delle conclusioni a cui sono arrivati, in un contraddittorio davanti al giudice.

[60] L'unico requisito indicato era una *general acceptance* (generale accettazione) della teoria scientifica che si voleva far valere nel processo, v. Court of Appeals of District of Columbia, «Frye v. United States, 293 F. 1013 (D.C. Cir. 1923)», 1923, https://web.archive.org/web/20230202073721/https://nij.ojp.gov/sites/g/files/xyckuh171/files/media/document/frye-v-US.pdf.

[61] Le *Federal Rules of Evidence* (norme federali in materia di prove) sono state adottate nel 1975, e permettono l'assunzione di qualsiasi prova, purché non sia vietata dalla legge, o sia irrilevante (*Rule 402*; cfr. art. 191 co. 1 e art. 190 co. 1 c.p.p.). Una prova è rilevante se permette di determinare se un certo fatto sia stato commesso o meno (*Rule 401*; cfr. art. 187 co. 1 c.p.p.). La *expert testimony* (perizia o consulenza tecnica) non menziona il requisito della *general acceptance* degli elementi scientifici che vengono menzionati dall'esperto (*Rule 702*), e non è nemmeno possibile aggiungerlo in via interpretativa, perché mentre è possibile usare le decisioni di *common law* (decisioni giurisprudenziali) per interpretare uno *statute* (legge), questa interpretazione non può essere *contra legem* (contraria alla legge), ed il criterio restrittivo in *Frye* contrasta con il *permissive backdrop* (spirito permissivo) che ispira le *Federal Rules of Evidence*. V. Supreme Court of the United States, «Daubert v. Merrell Dow Pharmaceuticals, Inc., 509 U.S. 579 (1993)», 1993, https://web.archive.org/web/20221012193634/https://tile.loc.gov/storage-services/service/ll/usrep/usrep509/usrep509579/usrep509579.pdf, pp. 585–589.

[62] *Ibidem*, p. 593.

scientifiche[63] e deve essere rilevante e utile per l'accertamento dei fatti.[64]

La seconda riguarda l'attendibilità della prova ritenuta ammissibile:[65]

- La teoria o tecnica scientifica deve essere verificabile, ed è preferibile che sia stata sottoposta a verificazione;[66]
- È preferibile (ma non necessario) che la teoria o tecnica sia stata pubblicata e soggetta a *peer-review*;[67]
- Il tasso di errore (conosciuto o potenziale) e l'esistenza e aggiornamento di standard che regolano la tecnica scientifica devono essere presi in considerazione;[68]
- La *general acceptance* menzionata in *Frye* continua ad essere rilevante, ma non è più l'unico fattore.[69]

In Italia, la sentenza Cozzini del 2010[70] introduce dei criteri analoghi a quelli previsti nel caso *Daubert*, che continuano ad essere riaffermati dalla Cassazione anche

[63]Pertanto, deve essere ottenuta mediante il metodo scientifico, e deve essere ragionevolmente certa. Nella scienza non esistono certezze assolute, ma il metodo scientifico permette di creare modelli sempre più precisi. In particolare, è importante che l'oggetto della deposizione deve essere stato soggetto a verifica empirica. V. Supreme Court of the United States, *op. cit.*, pp. 589–590.

[64]Cfr. gli artt. 190. co. 1 e 187 co. 1 c.p.p., v. Supreme Court of the United States, *ivi*, pp. 590–592.

[65]L'elenco indicato dalla corte non è tassativo, v. Supreme Court of the United States, *ivi*, p. 593.

[66]La possibilità di verificare, confutare o falsificare le teorie caratterizza la scienza, e la distingue dalle altre forme di ricerca e studio, v. Supreme Court of the United States, *ivi*, p. 593.

[67]La pubblicazione è un fattore importante, ma non determinante. Da sola la pubblicazione non è sufficiente a garantire l'affidabilità della teoria, e viceversa, teorie nuove (ma affidabili) potrebbero non essere state pubblicate. V. Supreme Court of the United States, *ivi*, pp. 593–594.

[68]V. Supreme Court of the United States, *ivi*, p. 594. Per quanto riguarda l'informatica forense, può essere difficile misurare il "tasso di errore" del software con precisione matematica, ed è molto più probabile che il giudizio abbia natura discorsiva. Inoltre, non ci si deve limitare a considerare solo gli standard più autorevoli, perché l'evoluzione tecnologica potrebbe averli già resi parzialmente obsoleti nel tempo necessario per la loro preparazione. In particolare, se l'oggetto dell'analisi è nuovo, si deve considerare anche la ricerca scientifica indipendente, che non è ancora stata formalizzata in standard.

[69]V. Supreme Court of the United States, *ivi*, p. 594. Ridurre l'ammissibilità solo al fatto che una teoria è largamente condivisa significa escludere l'ammissibilità di tutte le teorie che hanno solidi fondamenti scientifici, ma che semplicemente non hanno ancora ricevuto sufficiente attenzione dalla comunità scientifica. Questo è il caso tipico dell'informatica forense: dato che l'oggetto di studio è in continua evoluzione, sarebbe impossible riuscire ad ottenere un largo consenso su ogni singolo metodo di analisi.

[70]V. § 16 in Cassazione Penale, Quarta Sezione, «Sent. n. 43786/2010», 2010, https://web.archive.org/web/20211128212823/https://olympus.uniurb.it/index.php?option=com_content&view=article&id=3919:cassazione-penale-sez-4-13-dicembre-2010-n-43786&catid=17&Itemid=138.

di recente:[71]

> Al riguardo, mette conto richiamare i criteri di valutazione della prova scientifica delineati dalla giurisprudenza di legittimità, soprattutto sulla scorta dei cc.dd. canoni Daubert (dalla sentenza nordamericana Daubert vs Merrel Dow Pharmaceuticals, Inc. 509 U.S. 579, 113 S. Ct. 2786): «Per valutare l'attendibilità di una teoria occorre esaminare gli studi che la sorreggono. Le basi fattuali sui quali essi sono condotti. [L]'ampiezza, la rigorosità, l'oggettività della ricerca. Il grado di sostegno che i fatti accordano alla tesi. La discussione critica che ha accompagnato l'elaborazione dello studio, focalizzata sia sui fatti che mettono in discussione l'ipotesi sia sulle diverse opinioni che nel corso della discussione si sono formate. L'attitudine esplicativa dell'elaborazione teorica. Ancora, rileva il grado di consenso che la tesi raccoglie nella comunità scientifica. Infine, dal punto di vista del giudice, che risolve casi ed esamina conflitti aspri, è di preminente rilievo l'identità, l'autorità indiscussa, l'indipendenza del soggetto che gestisce la ricerca, le finalità per le quali si muove» (Sez. 4, n. 43786 del 17/09/2010, Cozzini, Rv. 248943-4).

In *Daubert* si afferma che l'uso di criteri più flessibili rispetto alla *general acceptance* non risulterà nell'ingresso di teorie pseudo-scientifiche nel processo, perché il processo accusatorio ha istituti per esaminare prove ammissibili, ma di dubbia affidabilità.[72]

Allo stesso modo, nella sentenza Knox del 2015 si afferma che per quanto riguarda la valutazione della prova "[l]e coordinate di riferimento dovranno essere quelle afferenti al principio del contraddittorio ed al controllo del giudice sul processo di formazione

[71]Cassazione Penale, Quinta Sezione, «Sent. n. 1801/2022», 2022, https://web.archive.org/web/20231222154506/https://www.italgiure.giustizia.it/xway/application/nif/clean/hc.dll?verbo=attach&db=snpen&id=./20220117/snpen@s50@a2022@n01801@tS.clean.pdf, p. 21.

[72]Ad esempio, l'esame incrociato e l'ammissione di prove contrarie. V. Supreme Court of the United States, *op. cit.*, pp. 595–596.

della prova".[73]

È possibile notare degli elementi in comune fra il diritto processuale ed il metodo scientifico: entrambi cercano di approssimare la realtà mediante un confronto fra più parti;[74] all'interno del confronto, si deve spiegare e difendere il proprio ragionamento;[75] infine, entrambi favoriscono la pubblicazione delle informazioni e la trasparenza nel procedimento.[76]

Dato che l'informatica forense ha una doppia natura, giuridica e scientifica, a maggior ragione deve essere ispirata da questi elementi; e dato che per l'esercizio dell'informatica forense è strettamente necessario usare del software, anche gli strumenti di analisi devono essere sviluppati ed utilizzati secondo i principi del confronto fra più parti, della motivazione del proprio operato, e della trasparenza.

2.2 Inquadramento legale e tecnico del software

2.2.1 Definizione di software libero

Il modello di software ideale per soddisfare le esigenze appena indicate è il software libero.[77] Il software può essere definito libero se sviluppato e distribuito al pubblico in

[73] Il "controllo del giudice" non va inteso come un ritorno al modello inquisitorio, ma come la necessità che il giudice valuti attivamente l'affidabilità della prova scientifica, invece di accettarla in maniera passiva. V. Cassazione Penale, Quinta Sezione, «Sent. n. 36080/2015», 2015, https://web.archive.org/web/20171104040843/https://www.giurisprudenzapenale.com/wp-content/uploads/2015/09/cass-pen-2015-36080.pdf, pp. 34–35.

[74] Nel contraddittorio, le parti cercano di approssimare la realtà storica criticando e cercando di contraddire gli argomenti dell'altra parte. Nella *peer-review*, gli scienziati cercano di approssimare il funzionamento dei fenomeni naturali creando e falsificando ipotesi e teorie.

[75] Le parti devono indicare in maniera dettagliata le teorie scientifiche seguite, le attività pratiche svolte, ed il processo logico che è stato seguito per arrivare ad una certa conclusione. La motivazione deve essere comprensibile, in modo che sia criticabile e falsificabile, e deve essere sufficientemente dettagliata, in modo che l'altra parte processuale o gli altri scienziati possano provare a ripetere l'esperimento e riprodurre i risultati.

[76] Nel diritto processuale l'obbligo del segreto cade dopo la conclusione delle indagini preliminari (art. 329 c.p.p.), le udienze sono pubbliche a pena di nullità (art. 471 co. 1 c.p.p.) e i casi in cui si procede a porte chiuse sono l'eccezione (art. 472 c.p.p.). Nel metodo scientifico, qualsiasi dettaglio o informazione che non viene pubblicata rende la *peer-review* più difficoltosa, e pertanto diminuisce la qualità della ricerca scientifica.

[77] In inglese *free software*, dove *free* va inteso come "libero da vincoli", e non "gratuito", v. Free Software Foundation, «What is Free Software?», 2023, https://web.archive.org/web/20231230224545/https:

maniera da garantire quattro libertà fondamentali:

> The freedom to run the program as you wish, for any purpose (freedom 0).
>
> The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is a precondition for this.
>
> The freedom to redistribute copies so you can help others (freedom 2).
>
> The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.[78]

Se anche solo una di queste libertà è limitata[79] si parla di software non-libero o proprietario.[80]

//www.gnu.org/philosophy/free-sw.en.html, sez. "Free software can be commercial".

[78] La libertà di eseguire il programma come si desidera, per qualsiasi scopo (libertà 0). La libertà di studiare come il programma funziona, e modificarlo in modo che funzioni a proprio piacimento (libertà 1). L'accesso al codice sorgente è una condizione necessaria per questa libertà. La libertà di ridistribuire copie così da aiutare gli alti (libertà 2). La libertà di modificare copie della tua versione modificata ad altri (libertà 3). Così facendo, puoi offrire all'intera comunità la possibilità di beneficiare delle tue modifiche. L'accesso al codice sorgente è una condizione necessaria per questa libertà.

[79] È irrilevante che la limitazione sia minima, ipotetica, difficile da far valere nella pratica, fondata su motivi etici, ecc. Ad esempio, la licenza di *jsmin*, scritto da Douglas Crockford, contiene la frase "*The Software shall be used for Good, not Evil.*" (il software dovrà essere usato per il bene, non per il male). Questa clausola è considerata una limitazione di una delle libertà fondamentali (v. Free Software Foundation, «What is Free Software?», cit, sez. "The freedom to run the program as you wish", e Free Software Foundation, «Various Licenses and Comments about Them», 2023, https://web.archive.org/web/20231018041504/https://www.gnu.org/licenses/license-list.html, sez. "The JSON License"), e pertanto rende il programma non-libero. V. R. Grove, *JSMin isn't welcome on Google Code*, 2009, https://web.archive.org/web/20230114224625/https://wonko.com/post/jsmin-isnt-welcome-on-google-code/.

[80] Free Software Foundation, «What is Free Software?», cit., sez. "The four essential freedoms".

2.2.2 Codice sorgente e codice macchina

L'esercizio delle libertà presuppone la possibilità di accedere al *source code* (codice sorgente). Il codice sorgente viene definito come:[81]

> *[T]he preferred form of the program for making changes in. Thus, whatever form a developer changes to develop the program is the source code of that developer's version.*[82]

È preferibile usare una interpretazione estensiva di questa definizione, che includa oltre al codice propriamente detto[83] anche qualsiasi altro file che sia necessario o utile per l'uso del programma,[84] o la cui presenza o contenuto altera il comportamento del programma in qualsiasi modo.[85]

Il codice sorgente si contrappone al *machine code* (codice macchina), la forma del programma che può essere eseguita direttamente dalla macchina.[86] Il codice sorgente viene trasformato in codice macchina da un programma chiamato *compiler* (compilatore).[87]

Il codice macchina presenta caratteristiche che ostacolano l'esercizio delle libertà

[81] *Ivi*, sez. "The freedom to study the source code and make changes".

[82] La rappresentazione del programma preferita per apportare cambiamenti. Pertanto, qualsiasi rappresentazione lo sviluppatore cambi al fine di sviluppare il programma è il codice sorgente di quella versione del programma.

[83] File di testo che contengono istruzioni scritte in un determinato linguaggio di programmazione.

[84] Ad esempio, file di configurazione, o file che contengono comandi per compilare ed installare il programma.

[85] Anche se il cambiamento riguarda solo come i dati vengono presentati all'utente, e non come vengono elaborati, quel file deve essere considerato parte del codice sorgente di quel programma.

[86] Per "macchina" si intende più precisamente il processore, che è in grado di comprendere solo un insieme di istruzioni limitato.

[87] Salvo bug nel compilatore, il codice sorgente ed il codice macchina rappresentano le stesse istruzioni, con la differenza che sono espresse in due linguaggi diversi. Il codice sorgente è scritto in un linguaggio più vicino ai linguaggi naturali, e quindi è più facile da leggere, scrivere e modificare per gli sviluppatori, ma non può essere eseguito. Viceversa, il codice macchina è un linguaggio che consiste solo in una lunga lista di istruzioni estremamente semplici, ed è difficile da modificare a mano, ma può essere eseguito dalla macchina.

fondamentali: non può essere sempre eseguito,[88] perde la struttura originale[89] e risulta "offuscato".[90]

2.2.3 Software e l.d.a.

Nell'ordinamento italiano, i programmi per elaboratore vengono espressamente equiparati ad un'opera letteraria (art. 1 co. 2 e art. 2 n. 8 l.d.a.),[91] indipendentemente dal fatto che il software sia espresso come codice sorgente o codice macchina.[92]

Questo inquadramento ha una serie di conseguenze. Alcune sono negative, perché sono funzionali ad ostacolare lo studio del software non-libero da parte dell'informatica forense.[93] Altre sono positive, perché (seppur entro certi limiti) permettono l'esercizio

[88]Il codice sorgente può sempre essere ricompilato e quindi funzionare su più processori. Questo limita la libertà di eseguire il programma. Il codice macchina consiste in un insieme chiuso di possibili istruzioni (*instruction set*) che vengono codificate in codice binario secondo un formato preciso. Questi elementi variano a seconda dell'architettura del processore (x86, AMD64, ARM, etc.), a seconda del momento di produzione del processore (i processori più nuovi supportano più istruzioni), ecc. Pertanto, il codice macchina è tendenzialmente legato all'architettura per cui è stato compilato.

[89]Questo limita la libertà di studiare e modificare il programma, perché diventa necessario ricostruire il suo funzionamento, e non è possibile fare riferimento al codice già esistente per apportare modifiche. Ad esempio, il codice sorgente può essere diviso in numerosi file, ma la compilazione risulta in un singolo file; informazioni come il nome delle variabili, dei valori nelle strutture di dati, delle funzioni, ecc., sono sostituite da riferimenti ad indirizzi di memoria, perché è l'unico formato che il processore comprende; i commenti che spiegano come il codice funziona vengono eliminati; ecc.

[90]Questo limita la libertà di studiare il programma, perché diventa difficile seguire il suo funzionamento. La perdita della struttura originale è già una prima forma di "offuscamento" del codice sorgente. Inoltre, il compilatore può riscrivere le istruzioni del codice macchina in modo da raggiungere lo stesso risultato, ma in maniera più efficiente (*compiler optimizations*), che rende ancora più difficile capire il funzionamento del codice. Ad esempio, un'istruzione usata per calcolare gli indirizzi di memoria può essere anche usata per compiere calcoli aritmetici (v. B. Visness, *Using the LEA instruction for arbitrary arithmetic*, 2022, https://web.archive.org/web/20220630160511/https://handmade.network/forums/articles/t/7111-using_the_lea_instruction_for_arbitrary_arithmetic). Ancora, è possibile introdurre del vero e proprio offuscamento intenzionale, ricombinando il codice in maniera da rendere estremamente difficile capire il suo funzionamento (ad esempio, v. xoreaxeaxeax, *movfuscator*, 2020, https://github.com/xoreaxeaxeax/movfuscator/tree/ea37dae93fbcd93f642c71a53878da588bd7ddb4).

[91]Legge 22 aprile 1941, n. 633, "Protezione del diritto d'autore e di altri diritti connessi al suo esercizio."

[92]Come indicato espressamente dall'art. 10(1) dell'accordo TRIPs (v. https://web.archive.org/web/20230929163013/https://biblioteche.cultura.gov.it/it/documenti/Servizio_III/4_accordo_trips_1994_x1x.pdf) e come desumibile dall'art. 4 del trattato OMPI sul diritto d'autore (v. https://web.archive.org/web/20231230130759/https://biblioteche.cultura.gov.it/it/documenti/Servizio_III/5_trattato_ompi_sul_diritto_d_autore_wct_1996_x1x.pdf).

[93]I software utilizzati dagli utenti tendono ad essere sistemi operativi e programmi proprietari. Da un punto di vista tecnico, l'analisi è già difficoltosa, ma il diritto d'autore permette l'aggiunta di ulteriori restrizioni, che complicano ulteriormente il lavoro dei tecnici. Ancora peggio, gli strumenti di analisi

delle libertà relative al software libero.

La conseguenza più immediata è Il "diritto esclusivo" dell'autore di creare copie dell'opera (software) previsto in generale dall'art. 13 l.d.a., e in maniera specifica per i "programmi per elaboratore" dall'art. 64-*bis* lett. *a* l.d.a.[94]

Una seconda conseguenza è la possibilità di limitare l'uso delle copie[95] mediante l'impiego di misure tecnologiche,[96] come previsto dall'art. 11 del trattato OMPI sul diritto d'autore[97] e dall'art. 102-*quater* l.d.a.:

> 1. I titolari di diritti d'autore [...] possono apporre sulle opere o sui materiali protetti misure tecnologiche di protezione efficaci che comprendono tutte le tecnologie, i dispositivi o i componenti che, nel normale corso del loro funzionamento, sono destinati a impedire o limitare atti non autorizzati dai titolari dei diritti.
>
> 2. Le misure tecnologiche di protezione sono considerate efficaci nel caso in cui l'uso dell'opera o del materiale protetto sia controllato dai titolari tramite l'applicazione di un dispositivo di accesso o di un procedimento di protezione, quale la cifratura, la distorsione o qualsiasi

usati dai tecnici tendono ad essere proprietari, e quindi senza l'accesso al codice sorgente è difficile, se non impossibile, spiegare come funzionano in maniera dettagliata all'interno del contraddittorio, e sapere se funzionano correttamente.

[94]Sono previste delle limitazioni ed eccezioni a questo diritto. Ad esempio, è possibile riprodurre opere "a fini di pubblica sicurezza, nelle procedure [...] giudiziarie" (art. 67 l.d.a.). Il processo penale potrebbe essere considerato una procedura penale con fini di sicurezza. Ancora, è possibile riprodurre le opere "per uso personale", purché non vengano distribuite al pubblico (art. 68 co. 1 e 6 l.d.a.).

[95]Si parla di "uso" e non "creazione" delle copie, perché è sempre possibile copiare i dati informatici. L'unico modo per evitare la creazione di copie non autorizzate è applicare dei meccanismi che impediscono l'esecuzione del software copiato senza autorizzazione. Ad esempio, si può controllare la presenza di *dongle* USB che contengono una licenza per l'uso del software in formato digitale, oppure si possono contattare i server dello sviluppatore per verificare che l'utente sia autorizzato ad usare il programma, ecc. L'efficacia di queste misure di protezione si basa sulle caratteristiche del software compilato: sono difficili da rimuovere perché è difficile capire quali parti del codice macchina contengono le istruzioni necessarie al loro funzionamento.

[96]La legge permette e protegge l'uso di misure tecnologiche che possono limitare la libertà di eseguire il programma. Spesso ci si riferisce all'uso di queste misure con l'acronimo *DRM* (*digital rights management*, gestione digitale dei diritti). Gli sviluppatori del software non sono tenuti ad usare questo tipo di misure (v. Jørgen Blomqvist, *Primer on International Copyright and Related Rights*, Edward Elgar Publishing, 2014, 207), ed in questo caso si parla di opere che sono *DRM-free* (libere da DRM).

[97]*Ibidem*, p. 205.

> altra trasformazione dell'opera o del materiale protetto, ovvero sia limitato mediante un meccanismo di controllo delle copie che realizzi l'obiettivo di protezione.

La l.d.a. non indica un elenco di *exceptions and limitations* ("eccezioni e limitazioni") che permettono di rimuovere o aggirare le misure di sicurezza.[98] Tuttavia, anche se la legge prevedesse esplicitamente questa possibilità, sarebbe preferibile evitare di rimuovere le misure di protezione per ragioni tecniche.[99]

Una terza conseguenza è la possibilità di studiare il funzionamento del programma, che viene data senza limitazioni significative ed è rafforzata dalla previsione di nullità per le clausole contrarie (art. 64-*ter* co. 3 l.d.a.):

> Chi ha il diritto di usare una copia del programma per elaboratore può, senza l'autorizzazione del titolare dei diritti, osservare, studiare o sottoporre a prova il funzionamento del programma, allo scopo di determinare le idee ed i principi su cui è basato ogni elemento del programma stesso, qualora egli compia tali atti durante operazioni di caricamento, visualizzazione, esecuzione, trasmissione o memorizzazione del programma che egli ha il diritto di eseguire. Le clausole contrattuali pattuite in violazione del presente comma e del comma 2 sono nulle.

L'ultima conseguenza è la possibilità di eseguire il *reverse-engineering* ("ingegneria a ritroso"),[100] ma per il solo fine dell'interoperabilità[101] (art. 64-*quater* co. 1 l.d.a.):

[98] L'art. 102-*quater* co. 3 afferma solo che resta salva la disciplina in generale sui programmi per elaboratore. Alcuni stati hanno ammesso la possibilità di rimuovere le misure di sicurezza, ma solo in casi limitati o eccezionali (ad esempio, per permettere l'uso da parte dell'autorità giudiziaria, o per ragioni di sicurezza nazionale). V. J. Blomqvist, *op. cit.*, p. 208.

[99] Per essere efficace, il DRM deve essere difficile da rimuovere, ma più il funzionamento del DRM è complesso, e quindi più è difficile sapere quali istruzioni nel codice macchina vanno rimosse, e maggiore è il rischio di andare a modificare anche il programma in maniera difficilmente imprevedibile. Questo margine di incertezza è inaccettabile nel software scientifico.

[100] Laddove la *software engineering* (ingegneria informatica) costruisce il software, la *reverse-engineering* serve a capire come il software è stato costruito. Consiste nell'analisi del codice macchina, già compilato, per studiare ed eventualmente cercare di ricostruire un'approssimazione del codice sorgente originale.

[101] L'interoperabilità è la "[c]apacità di due o più sistemi, reti, mezzi, applicazioni o componenti, di

> L'autorizzazione del titolare dei diritti non è richiesta qualora la riproduzione del codice del programma di elaboratore e la traduzione della sua forma ai sensi dell'art. 64-bis, lettere a) e b), compiute al fine di modificare la forma del codice, siano indispensabili per ottenere le informazioni necessarie per conseguire l'interoperabilità [...]

L'interoperabilità non può sconfinare nella creazione di "software sostanzialmente simile" (co. 2):

> Le disposizioni di cui al comma 1 non consentono che le informazioni ottenute in virtù della loro applicazione [...] siano utilizzate per lo sviluppo, la produzione o la commercializzazione di un programma per elaboratore sostanzialmente simile nella sua forma espressiva, o per ogni altra attività che violi il diritto di autore.

Infine, è di nuovo prevista la nullità per clausole contrattuali contrarie ai commi precedenti (co. 3).

Per quanto riguarda l'informatica forense, l'attività di analisi dei dati informatici rientra nella definizione di interoperabilità,[102] e non è "sostanzialmente simile" al programma originale.[103]

Gli artt. 64-*ter* e 64-*quater* l.d.a. sono estremamente importanti per l'informatica forense. Il primo permette di studiare il software proprietario, e quindi di creare un

scambiare informazioni tra loro e di essere poi in grado di utilizzarle." V. Treccani.it, *Interoperabilità*, 2008, https://web.archive.org/web/20231228151041/https://www.treccani.it/enciclopedia/interoperabilita_(Enciclopedia-della-Scienza-e-della-Tecnica)/.

[102]Normalmente, quando si parla di interoperabilità si pensa alla possibilità di aprire i file di Word (in formato DOC e DOCX) in programmi come LibreOffice, o di aprire i file di Photoshop (in formato PSD) in programmi come GIMP, ecc. In altre parole, si pensa all'uso di quei dati all'interno di programmi non specializzati, da parte di utenti ordinari. Tuttavia, non c'è una differenza significativa fra un utente che apre il file per visualizzarlo ed eventualmente modificarlo, ed un tecnico che lo apre per analizzarlo, in entrambi i casi è necessario che il programma sia in grado di leggere quel formato. Pertanto, si può comunque parlare di interoperabilità.

[103]I programmi di analisi spesso non hanno bisogno di modificare o scrivere i dati, ma solo di leggerli, perché è molto probabile che l'analisi venga svolta con algoritmi *ad hoc*, sviluppati in maniera indipendente rispetto al programma originale. Pertanto, è difficile parlare di una somiglianza sostanziale, quando le funzioni sono radicalmente diverse.

modello del suo funzionamento secondo il metodo scientifico.[104] Il secondo permette di creare strumenti di analisi che sono in grado di usare i formati proprietari.[105]

2.2.4 Licenze d'uso del software libero nell'ordinamento italiano

I contratti di licenza d'uso[106] del software sono un modello di origine statunitense in cui il licenziante (generalmente, chi sviluppa il software) concede al licenziatario (chi ne ottiene una copia) vari diritti, a titolo gratuito o oneroso.

Il software proprietario usa licenze *ad hoc*, che vietano (espressamente o implicitamente)[107] l'esercizio delle libertà fondamentali del software libero. È possibile limitare anche la libertà più fondamentale, la possibilità di eseguire il software (art. 64-*bis* co. 1 lett. *a*).[108]

Il software libero usa generalmente delle licenze standard, che possono essere chiamate *free software licenses*,[109] *open source licenses*,[110] o più genericamente, *FOSS licenses* (*free and open-source licenses*, licenze per software libero e a sorgente disponibile).[111]

[104]In sua assenza, l'attività di studio sarebbe una continua violazione del diritto d'autore.

[105]In sua assenza, sarebbe impossible sviluppare strumenti di analisi in grado di leggere i formati proprietari e non documentati pubblicamente (come ad esempio, il *filesystem* NTFS di Windows).

[106]Il termine "licenza" è un calco linguistico dall'inglese *license*. In italiano le "licenze" propriamente dette riguardano la possibilità di sfruttare economicamente il diritto di privativa industriale (come marchi e brevetti). V. Antonino Geraci, «I contratti di licenza d'uso del software», Università degli Studi di Parma, 2015, https://www.repository.unipr.it/handle/1889/2715, pp. 7–8. Nel seguito della trattazione, le espressioni "contratto di licenza d'uso" e "licenza" saranno usati in maniera intercambiabile.

[107]Per concedere l'uso dei diritti, è necessario indicarli espressamente e per iscritto (artt. 109 e 110 l.d.a.). Pertanto, non indicare che è possibile eseguire liberamente il software, crearne copie, modificarlo, ecc. è funzionalmente equivalente a negare espressamente queste facoltà.

[108]Ad esempio, imponendo dei limiti riguardo al tipo di hardware su cui il software può essere eseguito, sul numero di utilizzatori simultanei, ecc., e questi termini contrattuali possono essere fatti valere mediante l'uso di misure di sicurezza, che impediscono l'esecuzione del software in caso di loro violazione.

[109]Se rispettano la definizione di *free software*, v. Free Software Foundation, «Various Licenses and Comments about Them», cit.

[110]Se rispettano la definizione di *open source software* della OSI (v. Open Source Initiative, «The Open Source Definition», 2007, https://web.archive.org/web/20231231152615/https://opensource.org/osd/). Per un elenco, v. Open Source Initiative, «OSI Approved Licenses», 2024, https://web.archive.org/web/20240106034506/https://opensource.org/licenses/.

[111]In alcuni casi si usa l'espressione *FLOSS* invece che *FOSS*, per includere anche l'aggettivo *libre*. *Libre* ha lo stesso significato di *free*, ma non presenta la stessa ambiguità semantica.

Per quanto riguarda il software libero, è preferibile evitare di usare licenze meno conosciute,[112] o peggio ancora, scrivere una licenza originale. Piuttosto, è consigliabile usare le licenze FOSS più comuni,[113] per vari motivi: le licenze sono contratti, e quindi è opportuno che siano redatte da esperti,[114] le licenze più importanti sono state usate in procedimenti giudiziari,[115] e la compatibilità fra le licenze più utilizzate è già conosciuta.[116]

Le licenze FOSS possono essere utilizzate anche nell'ordinamento giuridico italiano. È preferibile considerare una licenza FOSS un contratto atipico,[117] e di evitare di forzarla negli schemi dei contratti tipici.[118] La licenza trasferisce dei diritti a chi riceve una copia del software[119] a titolo gratuito.[120]

[112]Ad esempio, è preferibile evitare licenze come la Unlicense o la WTFPL, e sostituirle con le licenze 0BSD o MIT. V. C. Morgan, *Don't use the Unlicense: it's an inferior license wrapped in an atrocious name.*, 2022, https://web.archive.org/web/20230519081106/https://chrismorgan.info/blog/unlicense/, e AA. VV., *Should I use the WTFPL for my FLOSS project?*, 2015, https://web.archive.org/web/20230131102031/https://opensource.stackexchange.com/questions/1359/should-i-use-the-wtfpl-for-my-floss-project.

[113]Ad esempio, le licenze GPL, MIT o Apache 2.0.

[114]Le licenze devono rispettare la legge, e devono essere interpretate per valutare quali effetti producono. È sempre preferibile che un contratto sia scritto da un esperto in materie legali, e faccia uso di clausole standard, in modo da minimizzare ambiguità o disaccordi nell'interpretazione.

[115]Questo significa che si è formato un precedente sulla validità ed interpretazione della licenza. Per una lista di casi riguardanti la GPL, v. AA. VV., *Have there been any lawsuits involving breach of open source licences?*, 2021, https://web.archive.org/web/20240110160950/https://opensource.stackexchange.com/questions/11452/have-there-been-any-lawsuits-involving-breach-of-open-source-licences.

[116]Uno dei vantaggi più importanti del software libero è la possibilità di condividere e riutilizzare il codice sorgente fra più progetti. A questo fine, è necessario verificare che le licenze con cui i componenti sono distribuiti siano compatibili fra di loro, e che i loro termini non si contraddicano. Questa analisi deve essere svolta una sola volta se tutti usano le stesse licenze, ma andrebbe fatta caso per caso se ciascuno usasse una propria licenza. V. Richard Stallman, «License Compatibility and Relicensing», 2021, https://www.gnu.org/licenses/license-compatibility.html.en.

[117]Il codice civile ammette la trasmissione di diritti anche mediante contratti atipici (art. 1322 c.c.).

[118]Ad esempio, in passato le licenze sono state qualificate come un contratto di locazione che ha ad oggetto l'uso del software (v. A. Geraci, *op. cit.*, p. 21), oppure come un contratto di compravendita che ha per oggetto la trasmissione del diritto di usare il software (v. A. Geraci, *ivi*, p. 23). Tuttavia, queste ricostruzioni presentano problemi. Ad esempio, locazione e compravendita riguardano solo l'uso del software (il software libero fornisce più diritti all'utente), sono contratti a titolo oneroso (il software libero di solito viene distribuito gratuitamente), non sono state pensate per il software (il software presenta caratteristiche particolari), ecc.

[119]I diritti che conseguono alla creazione dell'opera possono essere trasmessi mediante contratto, che deve avere forma scritta (artt. 107 co. 1 e 110 l.d.a). Si può intendere trasferito qualsiasi diritto contenuto nella l.d.a. il cui esercizio sia necessario per l'esercizio delle libertà che caratterizzano il software libero.

[120]Il software libero può essere usato per scopi commerciali, e può formare oggetto di vendita, ma le libertà associate al software libero non possono essere vendute. In altre parole, gli sviluppatori originali possono vendere una copia del programma, ma chiunque acquista quella copia è poi libero

2.2.5 Licenza GPL

La licenza *GNU GPL* (*GNU General Public License*) è una licenza *copyleft*[121] che strumentalizza il diritto d'autore per garantire le libertà previste dal software libero. La GPL afferma esplicitamente la possibilità di distribuire copie non modificate del programma:[122]

> *You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.*
>
> *You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.*

La sezione successiva regola la distribuzione di copie modificate:[123]

> *You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:*

di ridistribuirla gratuitamente; viceversa, se il codice è distribuito gratuitamente, ma l'esercizio delle libertà deve essere acquistato, non si può più parlare di codice libero. V. Free Software Foundation, «What is Free Software?», cit, sez. "Free software can be commercial").

[121] *Copyleft* è un gioco di parole con *copyright* (diritto d'autore). Normalmente il diritto d'autore serve a limitare la ridistribuzione dell'opera, mentre le licenze *copyleft* garantiscono questo diritto a chiunque ottenga una copia dell'opera.

[122] Le copie devono contenere l'indicazione degli autori e della licenza, l'indicazione della limitazione di responsabilità, e una copia della licenza. È possibile richiedere un pagamento per la copia, oppure offrire supporto o garanzie a pagamento. V. Free Software Foundation, «GNU General Public License, Version 3, 29 June 2007», 2007, https://www.gnu.org/licenses/gpl-3.0-standalone.html, sez. "4. Conveying Verbatim Copies.".

[123] In breve, devono essere rilasciate come codice sorgente, si deve indicare l'autore e la data delle modificazioni (lett. *a*), l'uso della licenza GPL (lett. *b*) e l'intera opera derivata deve essere rilasciata secondo i termini della stessa licenza (lett. *c*). V. Free Software Foundation, *ivi*, sez. "5. Conveying Modified Source Versions.".

a) *The work must carry prominent notices stating that you modified it, and giving a relevant date.*
b) *The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".*
c) *You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. [...] This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.*

Se un programma è composto di più parti si deve distinguere se sono indipendenti fra di loro, o se formano un unico programma, perché nel secondo caso la GPL si estende all'intero programma:

> *A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.*

La GPL permette la possibilità di fornire copie in forme diverse dal codice sorgente, ma si devono indicare le modalità per ottenere il codice sorgente.[124] L'obiettivo della GPL è di garantire che il software rimanga sempre libero, e che quindi il codice sorgente sia sempre disponibile agli utilizzatori.[125]

[124]Ad esempio, spesso il software libero viene distribuito come codice macchina, in modo che possa essere eseguito, senza dover essere compilato. Tuttavia, si deve offrire anche la possibilità di scaricare il codice sorgente, oppure di richiedere una copia. V. Free Software Foundation, *ivi*, sez. "6. Conveying Non-Source Forms.".

[125]Si contrappone alle licenze *permissive*, che invece ammettono la possibilità che il codice sorgente

Per raggiungere questo obiettivo la GPL contiene due "clausole virali": la prima è che qualsiasi opera derivata deve usare a sua volta la GPL,[126] la seconda è che qualsiasi opera che usa un componente già esistente e rilasciato con la GPL deve usare a sua volta la GPL.[127]

2.3 Confronto fra software proprietario e libero

2.3.1 Accesso al codice sorgente

Dopo aver descritto in dettaglio le esigenze del software per l'informatica forense, e le caratteristiche del software libero, è possibile dimostrare in maniera dettagliata perché il software libero è un modello migliore rispetto al software proprietario per quanto riguarda lo sviluppo di strumenti per il trattamento dei dati nell'informatica forense.

La caratteristica più importante del software libero è la possibilità di leggere il codice sorgente.[128] Gli svantaggi di non poter leggere il codice sorgente nel software

sia incluso all'interno di programmi proprietari, e non richiedono che l'utilizzatore del programma possa ottenere l'accesso al codice sorgente. V. A. Geraci, *op. cit.*, pp. 72–73.

[126] È un rapporto "verticale": se un programmatore modifica del codice distribuito con la GPL, deve usare la GPL anche per la versione modificata. V. A. Geraci, *ivi*, p. 71.

[127] Stallman (l'autore della GPL) ha chiarito che se un programma usa la libreria GPL *readline*, questo è sufficiente per rilasciare l'intero programma con la licenza GPL, anche se il programma e la libreria vengono distribuiti separatamente. Questa proprietà è estremamente utile per il software scientifico, perché garantisce che continui a rimanere software libero. V. Richard Stallman, Bruno Haible, «Why CLISP is under GPL», 2000, https://gitlab.com/gnu-clisp/clisp/-/blob/master/doc/Why-CLISP-is-under-GPL, sez. "From rms@gnu.ai.mit.edu Mon Oct 19 00:06:25 1992", "From rms@gnu.ai.mit.edu Fri Nov 6 21:31:33 1992" e "From rms@gnu.ai.mit.edu Sat Oct 31 01:29:01 1992".

[128] È importante far notare che se l'unica facoltà concessa è la possibilità di leggere il codice sorgente, ma si vieta la possibilità di modificarlo o distribuirlo, il software non è libero o *open source*, ma è proprietario, e più specificamente *source-available* (con codice sorgente disponibile). Ad esempio, il codice sorgente per il programma di backup Tarsnap è pubblicamente disponibile (v. https://github.com/Tarsnap/tarsnap/tree/dfcc22d1e19e6813841aa6bd731be3bb357b252f), ed è possibile scaricare e compilare il codice sorgente, ma è vietato compilare una versione modificata del software (v. Tarsnap.com, *Tarsnap Terms and Conditions*, 2024, https://web.archive.org/web/20240114184641/https://www.tarsnap.com/legal.html). Il motivo per cui non si permette di modificarlo è per garantire che il servizio costi il meno possibile (v. Tarsnap.com, *Whys of Tarsnap Terms and Conditions*, 2024, https://web.archive.org/web/20240329083745/http://www.tarsnap.com/legal-why.html, sez. "Why do people have to use unmodified Tarsnap client code?").

proprietario sono molteplici:

- Diventa più difficile rilevare e studiare i bug;[129]
- Non è possibile sapere se i metodi di analisi sono stati implementati correttamente;[130]
- Si limita l'esercizio del diritto alla difesa e lo svolgimento del contraddittorio;[131]
- Per il giudice diventa difficile motivare adeguatamente la sentenza.[132]

Il problema della motivazione della sentenza quando si usano software ed algoritmi proprietari è stato analizzato dalla giurisprudenza amministrativa, ma i principi valgono in generale:[133]

> Tra le indicate garanzie assume primaria importanza il rispetto del principio di trasparenza, che, com'è noto, trova un immediato corollario

[129]Dato che non è possibile confrontare come il programma si dovrebbe comportare (secondo il codice sorgente), e come si comporta nella pratica, può essere difficile sapere se un certo comportamento è voluto dagli sviluppatori, oppure è un bug, e se si trova un bug, non è possibile studiare il codice sorgente per trovare la sua origine, e cercare di capire come la sua presenza può aver influito sui risultati.

[130]Se il programma sta implementando dei metodi di analisi descritti dalla ricerca scientifica, non è possibile verificare che questi metodi siano stati implementati correttamente, ed il programma nel suo complesso funzioni correttamente. Ci si può solo fidare, o cercare di verificare l'implementazione con metodi indiretti (ad esempio, confrontando il comportamento del programma proprietario con altri programmi, di cui si conosce l'esatto funzionamento), ma entrambe le alternative creano margini di incertezza difficilmente accettabili.

[131]Dato che non è possibile sapere in dettaglio come lo strumento è giunto a quel risultato, diventa difficile riuscire a rispondere all'accusa in maniera specifica: sarebbe come ricevere una sentenza di condanna che non contiene la motivazione. Se non è possibile difendersi in maniera puntuale, diventa difficile avere un contraddittorio fruttuoso fra le parti. Si potrebbe ipotizzare una situazione in cui anche se il codice sorgente non è disponibile, il funzionamento del software proprietario è comunque ampiamente documentato, e quindi diventa possibile difendersi e avere un contraddittorio fruttuoso. Tuttavia, un conto è spiegare in maniera astratta come funziona il software, un conto è verificare le istruzioni contenute nel codice sorgente del programma che è stato usato nel caso concreto, e la loro corrispondenza al comportamento effettivo del programma quando viene eseguito; il diritto alla difesa ed il principio del contraddittorio sono soddisfatti solo nel secondo caso.

[132]Il giudice si trova davanti a due alternative. Può fidarsi dello strumento di analisi proprietario, usando argomenti come l'appello all'autorità ("è un programma usato da professionisti", "è sviluppato da esperti del settore", "se fosse inaffidabile non avrebbe così largo impiego", ecc.), oppure può non fidarsi, adducendo come motivo l'impossibilità di studiare il comportamento del programma. In entrambi i casi, si tratta di estremi, e non è possibile svolgere una discussione più articolata.

[133]TAR Campania, Napoli, Sez. III, «Sent. n. 7003/2022», 2022, https://web.archive.org/web/20231222125832/https://portali.giustizia-amministrativa.it/portale/pages/istituzionale/visualizza?nodeRef=&schema=tar_na&nrg=202105119&nomeFile=202207003_01.html&subDir=Provvedimenti.

nell'obbligo di motivazione degli atti amministrativi ex art. 3 l. 241/90 e che non può essere soppresso né ridotto sol per la presenza di un algoritmo all'interno dell'iter procedimentale.

Invero, il fatto che il provvedimento venga emanato sulla scorta di una complessa operazione di calcolo produce l'opposto effetto di rafforzare, per certi versi, l'obbligo motivazionale in capo all'Amministrazione, la quale dovrà rendere la propria decisione finale non solo conoscibile, ma anche comprensibile.

Occorre spostare l'attenzione a monte, sulla costruzione dell'algoritmo; su come i parametri dell'algoritmo vengono scelti (operazione di per sé soggettiva), e come si combinano tra loro; e ancor prima su come i termini assunti quale parametro siano stati realizzati.

La questione dell'individuazione dei termini da assumersi per la costruzione dell'algoritmo indica il momento in cui si opera la scelta caratterizzata da discrezionalità, sì che a queste fasi preliminari alla nascita dell'algoritmo devono essere anticipate le garanzie che devono accompagnare ogni scelta dell'amministrazione.

Fondamentale è a tal fine la garanzia di trasparenza, volte ad assicurare la conoscibilità della costruzione dell'algoritmo, anche, eventualmente, in funzione del sindacato sull'atto adottato sulla base dello stesso: "la decisione amministrativa automatizzata impone al giudice di valutare in primo luogo la correttezza del processo informatico in tutte le sue componenti: dalla sua costruzione, all'inserimento dei dati, alla loro validità, alla loro gestione" (cfr. Cons. St., sez. VI, n. 2270/2019).

In caso di decisione fondata su algoritmo, si richiede pertanto che sia assicurata una "declinazione rafforzata del principio di trasparenza", intesa come "piena conoscibilità della regola espressa in un linguaggio differente

da quello giuridico" (Cons. St., sez. VI, n. 2270/2019).

Negli Stati Uniti, la Corte Suprema del Winsconsin è giunta a conclusioni simili nel caso Loomis. Loomis aveva completato un questionario sul rischio di recidiva che usava un algoritmo proprietario e segreto, chiamato COMPAS, e sulla base del risultato di questo questionario, la corte "aveva condannato Loomis a sei anni di reclusione e cinque anni di *extended supervision*". Loomis impugnò questa sentenza, sostenendo che l'uso dell'algoritmo lo aveva privato del suo diritto ad avere una sentenza individualizzata, e nel 2016, la Corte Suprema affermò che l'algoritmo non può essere l'unico strumento usato per fondare una pronuncia di condanna.[134]

Se la conoscibilità del funzionamento di algoritmi e l'obbligo di motivazione del giudice vale per le decisioni prese dalla PA, a maggior ragione deve valere all'interno del processo penale, ed il miglior modo per garantire la conoscibilità è usare il software libero.

2.3.2 Libertà di riprodurre ed eseguire il programma

L'attività di ricerca nell'informatica forense deve essere soggetta a *peer review*. I dati informatici presentano delle caratteristiche particolari, che agevolano questa attività: possono essere copiati a costo nullo e con esattezza,[135] è possible creare una copia dell'intero sistema in cui l'analisi è stata svolta[136] ed è possibile automatizzare lo svolgimento di operazioni ripetitive mediante *script*.[137]

[134]Lucia Maldonato, «Algoritmi predittivi e discrezionalità del giudice: una nuova sfida per la giustizia penale», *Diritto Penale Contemporaneo*, fasc. 2, 2019, https://dpc-rivista-trimestrale.criminaljusticenetwork.eu/pdf/DPC_Riv_Trim_2_2019_maldonato.pdf: 404.

[135]Uno dei possibili ostacoli alla *peer review* è la necessità di riprodurre le condizioni dell'esperimento originale. Nel mondo materiale questo implica costi, e c'è un margine di errore ineliminabile; nel mondo digitale, copiare i dati non costa nulla, ed è possibile verificare l'esattezza della copia.

[136]Per funzionare, i programmi hanno bisogno di essere eseguiti all'interno di un sistema operativo. Per garantire la massima trasparenza riguardo le condizioni in cui l'esperimento è stato compiuto, e la piena ripetibilità dell'esperimento stesso, è utile fornire una copia forense dell'intero supporto che contiene il sistema operativo, programmi, configurazioni, ecc. che sono stati usati.

[137]Uno *script* consiste in una serie di istruzioni che vengono eseguite automaticamente, senza doverle scrivere a mano ogni volta. Questo permette di semplificare lo svolgimento di operazioni ripetitive e meccaniche (ad esempio, convertire i file in un determinato formato, preimpostare un programma,

Se gli strumenti di analisi sono proprietari, la possibilità di creare copie ed eseguirle incontra limitazioni legali e tecniche.[138] Viceversa, se gli strumenti di analisi sono software libero, chiunque può creare copie ed eseguirle a proprio piacimento, e questo agevola enormemente il processo di *peer review* nella fase di ricerca.

La possibilità di eseguire e distribuire il software sono utili anche per il contraddittorio nel processo, perché permettono a ciascuna parte di ottenere ed usare l'esatta copia degli strumenti di analisi usati dalla controparte,[139] e permettono di ripetere le analisi nel futuro.[140]

2.3.3 Libertà di modificare il programma

Il software proprietario è difficile da modificare ed estendere,[141] ed è molto meno resiliente rispetto al software libero.[142]

La libertà di modificare e distribuire copie modificate del software libero rileva in vari momenti: nella fase di *peer review*, i ricercatori possono pubblicare versioni

---

eliminare i file temporanei o intermedi generati dai programmi, ripristinare lo stato iniziale dell'ambiente di analisi, ecc.). Gli script possono anche svolgere i vari passi dell'attività di analisi (e quindi diventano anche una guida pratica riguardo l'uso del programma), e verificare la correttezza dei risultati (e quindi si automatizza il controllo della riproducibilità dell'esperimento). La *peer-review* si estende anche al contenuto di questi *script*.

[138] Ad esempio, non è possible creare una copia del sistema operativo Windows, non solo per ragioni legali, ma anche perché la licenza per utilizzarlo è legata all'hardware del computer. Se l'hardware cambia in maniera significativa (perché l'ambiente di analisi è stato copiato su un'altra macchina, che contiene hardware diverso), sarà necessario attivare di nuovo il sistema operativo. V. Microsoft, *Reactivating Windows after a hardware change*, 2023, https://web.archive.org/web/20231213170756/https://support.microsoft.com/en-us/windows/reactivating-windows-after-a-hardware-change-2c0e962a-f04c-145b-6ead-fb3fc72b6665. Ragionamenti analoghi possono essere svolti per gli strumenti di analisi proprietari che richiedono *dongle* USB per il loro funzionamento.

[139] Il contraddittorio può essere visto come una sorta di *peer review* all'interno del processo. Anche se le finalità sono diverse (nella *peer review* si cerca di falsificare una teoria, nel contraddittorio si cerca di difendere la propria posizione), le modalità sono simili (in entrambi i casi si cercano i punti deboli e si muovono critiche verso l'operato altrui).

[140] La copia del software può essere conservata insieme alla copia dei dati.

[141] Specialmente se non viene fornito il codice sorgente.

[142] Si parla del *maintainer hit by a bus problem* (problema dello sviluppatore colpito da un autobus). La possibilità (legale e materiale) di modificare e aggiornare il software proprietario è nelle mani di poche persone, e pertanto il software proprietario è estremamente fragile, specie se confrontato con il software libero.

corrette o migliorate del software,[143] nella fase di sviluppo del software, è possibile offrire *patch*[144] o modificare il software già esistente per adattarlo alle esigenze dell'informatica forense;[145] e nella fase di analisi, è possibile modificare il software per gestire le esigenze contingenti.[146]

Il software libero permette anche la creazione di *fork* (bivi).[147] I *fork* vengono generalmente creati per tre motivi: a seguito di disaccordi fra gli sviluppatori,[148] per esplorare nuove soluzioni,[149] o perché il progetto originale non è più aggiornato.[150]

2.3.4 Altre caratteristiche

È possibile confrontare il software proprietario ed il software libero sulla base di altre caratteristiche:

Per quanto riguarda le modalità di finanziamento, il software proprietario è generalmente offerto a pagamento,[151] ed è sviluppato per ottenere un profitto. Per

[143] Come supplemento pratico alle critiche o osservazioni teoriche.

[144] Se qualcuno copia il codice sorgente, e apporta modifiche (miglioramenti, correzioni di bug, ecc.), può creare una *patch* (pezza), un file che contiene le istruzioni per applicare queste modifiche al codice originale. La *patch* può essere condivisa con gli sviluppatori originali, che possono decidere di integrarla nel codice originale, in modo che tutti possano beneficiarne.

[145] Ad esempio, le distribuzioni GNU/Linux sono sistemi operativi liberi, e sono state modificate per creare distribuzioni specializzate per l'informatica forense.

[146] Ad esempio, è possibile correggere bug nella propria copia del software, o modificare gli strumenti di analisi già esistenti, o addirittura aggiungerne di nuovi, se necessario per la risoluzione dei quesiti posti dal giudice. Tutte queste modifiche dovranno essere oggetto di contraddittorio.

[147] Un *fork* è una copia del programma, che viene sviluppata in maniera indipendente rispetto al progetto originale. Il nome deriva dal fatto che il codice sorgente originale inizia a divergere in due direzioni separate.

[148] I disaccordi possono riguardare questioni puramente tecniche, oppure anche questioni relative alle modalità di sviluppo del progetto. Ad esempio, se un software è sviluppato esclusivamente da una sola persona, che ha una visione del software estremamente specifica, e non accetta contribuzioni di terze parti, è possibile fare un *fork* di quel software, in modo che la partecipazione allo sviluppo del *fork* sia più democratica.

[149] Ad esempio, Rekall era nato come un *fork* di Volatility per valutare la possibilità di rendere il codice più modulare, e migliorare le prestazioni e la facilità d'uso, ma successivamente è stato abbandonato, v. https://github.com/google/rekall/tree/55d1925f2df9759a989b35271b4fa48fc54a1c86.

[150] Lo sviluppo del software libero può essere ripreso e continuato in qualsiasi momento da chiunque abbia le capacità tecniche per farlo, è possibile adattare ed aggiornare il software, in maniera che continui a funzionare anche sui sistemi più recenti.

[151] Dato che il mercato è ristretto, ed è destinato ai professionisti, le licenze per il software di analisi forense costano centinaia o migliaia di euro. Pertanto, gli sviluppatori hanno una maggiore disponibilità economica da reinvestire nello sviluppo del software.

aumentare i profitti è possible ridurre i tempi e costi di sviluppo, rischiando di introdurre *technical debt*,[152] oppure aumentare le vendite, falsando la percezione del software da parte degli utenti.[153]

Il software libero di solito non viene venduto[154] ma il suo sviluppo può essere finanziato in vari modi.[155]

Per quanto riguarda l'uso di tecnologie coperte da segreti industriali o brevetti, il software proprietario può facilmente fare uso di queste tecnologie.[156]

Il software libero può usare tecniche di *reverse-engineering* per i segreti industriali, e distribuire solo il codice sorgente per le tecnologie coperte da brevetto.[157]

[152]Il *technical debt* (debito tecnico) consiste nel costo (in termini di produttività futura persa) che si accumula quando i programmatori scrivono codice senza considerare quanto sarà difficile estenderlo e mantenerlo nel futuro. Si può fare un'analogia con il comprare il veicolo più economico che si trova. Anche se nel breve termine si risparmia, nel lungo termine l'inaffidabilità del mezzo ed il costo delle riparazioni nullificheranno il risparmio iniziale. Esistono numerose cause che portano al *technical debt*, come la duplicazione del codice, non scrivere commenti, eseguire pochi *test* di funzionamento, non semplificare le parti che contengono codice complesso, non usare una guida di stile per il codice, ecc; V. J.L. Letouzey, D. Whelan, *Introduction to the Technical Debt Concept*, n.d., https://web.archive.org/web/20200707112025/https://www.agilealliance.org/wp-content/uploads/2016/05/IntroductiontotheTechnicalDebtConcept-V-02.pdf.

[153]Pertanto, i programmatori daranno maggiore priorità alla quantità di funzioni piuttosto che la loro qualità, e si nasconderanno o minimizzeranno i difetti del software, in modo da farlo sembrare migliore di quanto effettivamente sia. In ogni caso, l'impossibilità di accedere al codice sorgente rende difficile valutare l'effettiva qualità del software.

[154]L'incentivo al suo sviluppo è la creazione di software utile per sé e per gli altri, e pertanto c'è anche un incentivo a far funzionare il software correttamente.

[155]Mediante donazioni occasionali o ricorrenti (ad esempio, RPCS3, un progetto per l'emulazione della console PlayStation 3, ottiene circa 2.000 euro al mese da più di 500 donatori, v. https://web.archive.org/web/20231127232724/https://www.patreon.com/nekotekina/about), mediante *bounties* ("taglie", ossia somme offerte agli sviluppatori, in cambio dell'aggiunta di funzionalità), vendendo eccezioni alle licenze libere (in modo che il software possa essere usato in un prodotto proprietario, v. Richard Stallman, «Selling Exceptions to the GNU GPL», 2021, https://www.gnu.org/philosophy/selling-exceptions.html), con contratti di sponsorizzazione (ad esempio, v. https://web.archive.org/web/20240109144332/https://curl.se/sponsors.html), offrendo servizi di assistenza tecnica e *consulting* a pagamento, ecc.

[156]Ad esempio, il *filesystem* NTFS, usato da Windows, non è documentato pubblicamente, e per poterlo analizzare è necessario ottenere una specifica tecnica. È probabile che la Microsoft sia disposta a fornire la documentazione necessaria a sviluppatori di software di analisi proprietario. Il segreto industriale rimane protetto da un punto di vista legale con *non-disclosure agreements* (accordi di confidenzialità), e da un punto di vista tecnico dalla compilazione e dalla segretezza del codice sorgente. Viceversa, nel software libero è impossibile mantenere segreti industriali. Per quanto riguarda i brevetti, il costo per il loro uso può essere trasferito su chi acquista una copia del software proprietario.

[157]Ad esempio, la tecnica di compressione audio MP3 è stata protetta da un brevetto fino al 2017 (v. A. Orlowski, *MP3 'died' and nobody noticed: Key patents expire on golden oldie tech*, 2017, https://web.archive.org/web/20200701131247/https://www.theregister.com/2017/05/16/mp3_dies_nobody_noticed/), ma il progetto *LAME* per la codifica e decodifica di file MP3 è sempre stato distribuito (e continua ad essere

Per quanto riguarda la trasparenza ed imparzialità, il software proprietario non è sviluppato in maniera trasparente,[158] e tende a soddisfare gli interessi, desideri e necessità di un numero ristretto di persone.[159]

Il software libero spesso viene sviluppato in maniera trasparente,[160] e nel tempo, tende naturalmente a supportare quanti più formati e funzionalità possible.[161]

Per quanto riguarda la difficoltà d'uso, il software proprietario ha maggiori risorse economiche, quindi può investire nella creazione di interfacce grafiche intuitive, e nella creazione di manuali dettagliati.

Il software libero tende a preferire programmi a riga di comando,[162] e l'eventuale mancanza di documentazione è comunque controbilanciata dalla possibilità di consultare il codice sorgente.

2.3.5 Impossibilità di usare il software libero per i captatori

Quanto detto finora ha dimostrato che il modello del software libero è il modello preferibile per l'informatica forense, ma esiste almeno un caso in cui usare il software libero non è possibile, perché andrebbe a ledere l'efficacia delle indagini.

I captatori sono virus informatici usati dall'autorità giudiziaria a fini investigativi, capaci di compiere un numero enorme di operazioni, che vanno ad incidere su vari diritti fondamentali della persona.[163] Questi diritti sono "inviolabili", e possono

distribuito, anche dopo la scadenza del brevetto) esclusivamente come codice sorgente, e non come file binario (v. https://web.archive.org/web/20210904101433/https://lame.sourceforge.io/lame-faq.en.php, sez. "Tell me the history of LAME.").

[158]Ci sono pochi incentivi a pubblicare il codice sorgente, discutere apertamente i bug, mostrare le comunicazioni tra sviluppatori, ecc.

[159]Generalmente, gli sviluppatori originali, oppure chi finanzia il suo sviluppo.

[160]In generale, dato che il codice è già pubblico, c'è un incentivo a rendere anche altre informazioni pubbliche.

[161]Spesso il software libero riesce a supportare anche i formati proprietari, a seguito di *reverse-engineering*.

[162]La mancanza di interfacce grafiche non è uno svantaggio, è semplicemente un paradigma diverso, e non necessariamente inferiore. Infatti, l'uso di programmi a riga di comando permette di automatizzare le operazioni con facilità, scrivendole all'interno di file di testo, e quindi di creare metodi di analisi facilmente ripetibili. Ancora, se le interfacce nei programmi proprietari sono particolarmente intuitive, probabilmente stanno nascondendo opzioni o informazioni importanti all'utilizzatore.

[163]La libertà personale (art. 13 Cost.), intesa in senso estensivo come il libero sviluppo della persona

essere limitati solo nei casi e modi indicati dalla legge, ma la legge non disciplina adeguatamente l'uso di questi strumenti.[164]

I captatori permettono anche un'attività che viene chiamata "perquisizione on-line", che non può essere ricondotta alla perquisizione propriamente detta (art. 247 c.p.p.),[165] e pertanto viene considerata una prova atipica (art. 189 c.p.p.).[166]

L'art. 89 disp. att. c.p.p. prevede che i captatori devono essere "programmi conformi ai requisiti tecnici stabiliti con decreto del Ministro della giustizia".[167] I requisiti tecnici sono specificati all'art. 4 del decreto, che ha una formulazione piuttosto generica:[168]

1. I programmi informatici funzionali all'esecuzione delle intercettazioni mediante captatore informatico su dispositivo elettronico portatile sono elaborati in modo da assicurare integrità, sicurezza e autenticità dei dati captati su tutti i canali di trasmissione riferibili al captatore.
2. I sistemi di sicurezza adottati a norma del comma 1 consentono che solo gli operatori autorizzati abbiano accesso agli strumenti di comando e funzionamento del captatore.

---

umana; il domicilio (art. 14 Cost.), inteso anche nel senso di "domicilio informatico"; la riservatezza delle comunicazioni (art. 15 Cost.), che per la formulazione ampia data dalla Costituzione include anche le comunicazioni in formato digitale. V. Gaia Caneschi, «Le nuove indagini tecnologiche e la tutela dei diritti fondamentali. L'esperienza del captatore informatico», *Diritto Penale Contemporaneo*, fasc. 2, 2019, https://dpc-rivista-trimestrale.criminaljusticenetwork.eu/pdf/DPC_Riv_Trim_2_2019_caneschi.pdf, pp. 417–429, pp. 418–420.

[164]Il codice di procedura penale disciplina espressamente solo l'intercettazione di comunicazione fra presenti (art. 266 co. 2 e 2-*bis*), ponendo alcune limitazioni ai luoghi in cui può essere compiuta, e l'intercettazione di flussi di dati informatici fra più sistemi (art. 266-*bis*).

[165]Il motivo è che mancano le garanzie informative e difensive tipiche della perquisizione, e mentre la perquisizione riguarda solo la ricerca del corpo del reato, la perquisizione on-line può riguardare l'intero contenuto del dispositivo. V. G. Caneschi, *op. cit.*, p. 421.

[166]Più in generale, qualsiasi altra attività che può essere svolta con un captatore, ma che non rientra nelle prove tipiche, è una prova atipica. Le prove atipiche devono essere idonee ad accertare i fatti, non devono pregiudicare la libertà morale della persona, ed il giudice deve sentire le parti sulla modalità di assunzione della prova. V. G. Caneschi, *ivi*, pp. 421–422.

[167]Decreto del Ministero della Giustizia del 20 aprile 2018, "Disposizioni di attuazione per le intercettazionii [sic] mediante inserimento di captatore informatico e per l'accesso all'archivio informatico a norma dell'articolo 7, commi 1 e 3, del d.lgs. 216/2017", v. https://web.archive.org/web/20240402130505/https://www.giustizia.it/giustizia/it/mg_1_8_1.page?contentId=SDC289658.

[168]G. Caneschi, *op. cit.*, p. 425.

3. I medesimi sistemi di sicurezza prevedono:
    a) misure di offuscamento o evasione per impedire l'identificazione del captatore e dei dati captati, sia da parte di operatori umani, che per mezzo di specifico software;
    b) misure idonee ad assicurare la permanenza e l'efficacia del captatore sul dispositivo durante tutto il periodo di attività autorizzata e con i limiti previsti dal provvedimento autorizzativo, in modo da garantire il completo controllo da remoto.
4. I programmi informatici funzionali all'esecuzione delle intercettazioni mediante captatore consentono la trasmissione di tutte le informazioni necessarie a definire il contesto dell'acquisizione.
5. I programmi informatici sono periodicamente adeguati a standard di funzionalità ed operatività in linea con l'evoluzione tecnologica.

L'impossibilità di usare il software libero per i captatori deriva esclusivamente dal terzo comma.[169] Dato che il captatore è un virus informatico, non può rivelare come funziona.[170] Il captatore si trova in una situazione paradossale, per cui può essere necessario usarlo per la prova di determinati reati,[171] ma allo stesso tempo, il suo uso produce risultati intrinsecamente poco affidabili.[172]

[169]Le libertà previste dal software libero permetterebbero di controllare il rispetto dei requisiti contenuti negli altri commi, e agevolerebbe l'ammissione della prova atipica (art. 189 c.p.p.), perché è possibile dimostrare la sua idoneità ad accertare i fatti, e presentare il suo esatto meccanismo di funzionamento al giudice.

[170]Altrimenti, gli sviluppatori del sistema operativo pubblicherebbero un aggiornamento per motivi di sicurezza, ed il captatore diventerebbe inutilizzabile. Ad esempio, la Apple offre ricompense (da qualche migliaio di dollari, fino a milioni di dollari, a seconda della gravità) a chiunque trovi delle vulnerabilità nel loro sistema operativo. V. Apple.com, *Apple Security Bounty Categories*, 2023, https://web.archive.org/web/20231127163613/https://security.apple.com/bounty/categories/.

[171]Ad esempio, i reati elencati all'art. 266 co. 1 lett. f c.p.p. sono difficili da provare senza l'uso di intercettazioni.

[172]Anche volendo, il captatore non potrebbe mai essere software libero.

Capitolo 3

Sviluppo di software scientifico libero

3.1 Fattori di valutazione del software

3.1.1 Rilevanza per i giuristi

Il capitolo precedente ha dimostrato che da un punto di vista teorico, il modello del software libero è il modello di sviluppo che soddisfa maggiormente le esigenze dell'informatica forense.

Questo capitolo si concentra sugli aspetti pratici e tecnici dello sviluppo del software. In primo luogo è importante indicare ai giuristi quali sono gli elementi tecnici utili da menzionare nel contraddittorio per sostenere l'affidabilità del proprio strumento di analisi, e screditare lo strumento dell'altra parte.[1]

[1] In particolare, è possibile screditare il software proprietario per il solo fatto che non può essere sottoposto allo stesso livello di scrutinio del software libero. Questo non significa che il software libero sia necessariamente libero da difetti, o migliore del software proprietario, ma piuttosto, che è più facile rilevare questi difetti, e che è possibile sapere esattamente come e perché il software funziona.

3.1.2 Linguaggio di programmazione

Il primo fattore da valutare, e il più importante, è il linguaggio di programmazione in cui il software è stato scritto. Ogni linguaggio di programmazione ha caratteristiche diverse, che possono essere divise in due categorie: caratteristiche che aiutano la creazione di software ad uso scientifico,[2] e caratteristiche che rendono più difficile creare programmi che funzionano correttamente.

È preferibile usare:

- Linguaggi *memory-safe*[3] rispetto a linguaggi *memory-unsafe*;[4]
- Linguaggi *statically-typed* e *strongly-typed*,[5] piuttosto che linguaggi *dynamically-typed*;[6]
- Linguaggi che usano gli *out-of-band errors*;[7]

[2]Idealmente, i linguaggi di programmazione dovrebbero tendere verso il "pozzo del successo", ossia, dovrebbero rendere facile seguire le *best practices*, ma soprattutto, dovrebbero rendere difficile non seguirle. V. J. Atwood, *Falling Into The Pit of Success*, 2007, https://web.archive.org/web/20140402064217/https://blog.codinghorror.com/falling-into-the-pit-of-success/.

[3]Nei linguaggi *memory-safe*, la gestione della memoria è completamente automatica. V. Manuele Pasini, «Programmazione memory-safe senza garbage collection: il caso del linguaggio Rust», *Alma Mater Studiorum – Università di Bologna*, 2019, p. 21.

[4]La gestione manuale della memoria nei linguaggi *memory-unsafe* è la causa della maggioranza dei bug. V. A. Gaynor, *Introduction to Memory Unsafety for VPs of Engineering*, 2019, https://web.archive.org/web/20190812151808/https://alexgaynor.net/2019/aug/12/introduction-to-memory-unsafety-for-vps-of-engineering/ e P. Kehrer, *Memory Unsafety in Apple's Operating Systems*, 2019, https://web.archive.org/web/20190725163137/https://langui.sh/2019/07/23/apple-memory-safety/.

[5]Ossia, il programmatore deve indicare esplicitamente che tipo di valori che possono essere usati all'interno del programma e come questi valori devono essere trasformati. V. T. Hurd, *Introduction to Static, Dynamic, Strong and Weak Data Types*, 2021, https://web.archive.org/web/20210603180908/https://www.sitepoint.com/typing-versus-dynamic-typing/.

[6]I linguaggi *dynamically-typed* sono più facili da sviluppare, e richiedono meno codice, ma il prezzo da pagare è che il controllo sul corretto uso dei tipi avviene solo quando il programma è in fase di esecuzione. Nei linguaggi *statically-typed*, un programma che contiene errori nell'uso dei tipi non potrà nemmeno essere avviato. V. la sez. "Typing" in John K. Ousterhout, «Scripting: Higher Level Programming for the 21st Century», 1998, https://ieeexplore.ieee.org/document/660187, p. 24.

[7]Alcuni linguaggi tendono ad indicare gli errori *in-band*, e restituiscono un singolo risultato, che può essere il risultato dell'operazione, oppure un valore particolare che indica la presenza di un errore. È più difficile verificare la presenza di errori, e le informazioni diagnostiche sono più scarne. V. F. Long, W. Snavely, *ERR52-J. Avoid in-band error indicators*, 2017, https://web.archive.org/web/20230329034143/https://wiki.sei.cmu.edu/confluence/display/java/ERR52-J.+Avoid+in-band+error+indicators. È possibile cercare di approssimare un errore *out-of-band* restituendo un singolo valore che contiene al suo interno più elementi, ma è necessario che tutto il codice si adegui a questa nuova convenzione. Ad es., v. Team di Sviluppo GTK, *Error reporting*, 2021, https://web.archive.org/web/20210921180746/https://docs.gtk.org/glib/error-reporting.html.

- Linguaggi che sono maturi[8] e largamente utilizzati;[9]
- Linguaggi che sono semplici e *opinionated* rispetto a linguaggi complessi o *unopinionated*.[10]

Nel caso in cui sia necessario usare dei linguaggi che presentano delle caratteristiche sfavorevoli, è importante che gli sviluppatori cerchino di mitigare i loro effetti e documentino il loro operato, in modo che i tecnici possano argomentare che il software sia comunque affidabile.[11]

3.1.3 Documentazione del codice

La documentazione è importante per qualsiasi tipo di software, ma è una necessità imprescindibile per il software ad uso scientifico. Il termine "documentazione" è un

[8] È preferibile evitare linguaggi nuovi, sperimentali, o che cambiano di frequente, e concentrarsi su linguaggi che esistono da tempo, maturi e stabili. Ad esempio, C/C++, Java e Python sono stati creati decenni fa, sono largamente usati dall'industria. È presumibile che continueranno ad essere supportati ed utilizzati nel tempo, senza cambiamenti significativi. Il linguaggio Go è più recente, ma gli sviluppatori hanno promesso che si impegneranno a garantire la retro-compatibilità delle versioni successive del linguaggio con le versioni precedenti. V. R. Cox, *Backward Compatibility, Go 1.21, and Go 2*, 2023, https://web.archive.org/web/20230814162240/https://go.dev/blog/compat.

[9] Più un linguaggio è largamente utilizzato, e più è facile trovare risorse tecniche, codice da riutilizzare, programmatori che possono aiutare a migliorare il codice, ecc. Ad esempio, i linguaggi C e C++ sono rischiosi da utilizzare nel software scientifico perché *memory-unsafe*, ma allo stesso tempo, hanno larghissima utilizzazione nel mondo della programmazione, e quindi esiste un grande numero di risorse e strumenti che aiutano a sviluppare applicazioni robuste ed affidabili.

[10] Se un linguaggio è flessibile, e offre più modi per risolvere lo stesso problema, programmatori diversi useranno modi diversi, andando a violare il *principle of least surprise* (principio della sorpresa minima), e complicando la comprensione e manutenzione del codice. Ad esempio, basta confrontare il motto del linguaggio Python, *There should be one – and preferably only one – obvious way to do it*, con il motto del linguaggio Perl, *There's more than one way to do it*. Per una discussione dei rischi che un linguaggio particolarmente flessibile pone, v. R. Winestock, *The Lisp Curse*, 2011, https://web.archive.org/web/20110416211304/http://winestockwebdesign.com/Essays/Lisp_Curse.html.

[11] Ad esempio, i programmi che interagiscono direttamente con i componenti più basilari del sistema operativo (*filesystem*, memoria RAM, interfacce di rete, ecc.) devono essere scritti in linguaggi di basso livello, come C, che però tendono ad essere *memory-unsafe*. Pertanto, è utile usare Valgrind (v. https://web.archive.org/web/20231113151236/http://valgrind.org/docs/manual/mc-manual.html) per rilevare i bug relativi alla memoria. Viceversa, i programmi per analizzare i dati tendono ad essere scritti in linguaggi di alto livello, che però tendono ad essere *dynamically-typed*. Pertanto, è utile usare strumenti come MyPy (v. https://mypy-lang.org/) per controllare il corretto uso dei tipi. Più in generale, è utile seguire le linee-guida previste per il codice che sarà utilizzato in applicazioni critiche, come i sistemi aerospaziali, v. Gerard J. Holzmann, «The Power of 10: Rules for Developing Safety-Critical Code», 2006, https://ieeexplore.ieee.org/document/1642624.

termine generico: esistono più forme di documentazione, che si differenziano per il loro livello di astrazione[12] e per la loro finalità.[13]

Così come il codice, anche la documentazione può contenere *bug*[14] ed è soggetta ad una licenza, che può essere a sua volta libera.[15]

Si possono distinguere due tipi di documentazione. La prima è la documentazione che interessa principalmente agli sviluppatori:

- I commenti formano parte integrante del codice. Il loro scopo è di chiarire quanto non è immediatamente evidente da una semplice lettura del codice.[16] Nel codice ad uso scientifico, la presenza di commenti è un aiuto fondamentale per comprendere a pieno l'organizzazione ed il funzionamento del software;
- Il *reference manual* (manuale di riferimento) spiega nel dettaglio ogni componente del programma, e permette di studiare il funzionamento del programma senza dover leggere ogni riga del codice sorgente. Esempi di *reference manuals* sono il manuale per la libreria C GNU,[17] e la documentazione

[12]Ossia, quanto sono vicine alle singole istruzioni di cui il codice è composto.

[13]In linea generale, la documentazione può servire a spiegare il "perché" il programma è stato progettato in un certo modo, o il "come" funziona e trasforma i dati.

[14]Si parla di bug della documentazione se non è chiara o completa (perché rende difficile capire se il programma risponde alle proprie esigenze, o se si sta usando il programma correttamente) o se è difforme rispetto al comportamento del programma (perché è difficile determinare se il programma produce risultati incorretti, oppure la documentazione è incorretta). V. sez. "Reporting Bugs" in Free Software Foundation, «The GNU C Library Reference Manual, for version 2.38», 2023, https://web.archive.org/web/20231227043035/https://sourceware.org/glibc/manual/pdf/libc.pdf, p. 1121.

[15]I criteri per determinare se la documentazione è libera sono la possibilità di ridistribuire copie della documentazione insieme alle copie del codice, e la possibilità di modificare la documentazione, in modo che rifletta le eventuali modifiche apportate al codice. V. Free Software Foundation, *ivi*, p. 1149

[16]In altre parole, devono realmente essere un "commento" al codice, e non una semplice "traduzione" del linguaggio di programmazione in un linguaggio naturale. Per una lista di buone pratiche relative ai commenti, v. E. Spertus, *Best practices for writing code comments*, 2021, https://web.archive.org/web/20211223145454/https://stackoverflow.blog/2021/12/23/best-practices-for-writing-code-comments/. In linea generale, tra scrivere codice complesso e spiegarlo con commenti, e scrivere codice semplice che non ha bisogno di commenti, è sempre preferibile la seconda opzione; v. D. Orr, *Write code for humans. Design data for machines.*, 2020, https://web.archive.org/web/20200402061509/https://douglasorr.github.io/2020-03-data-for-machines/article.html. Nel caso in cui sia assolutamente necessario scrivere codice complesso, e cercare di semplificarlo è inutile o impossibile, anche questo va indicato nei commenti; ad es., v. N. Muller, *XeePhotoshopLoader.m*, 2013, https://github.com/zepouet/Xee-xCode-4.5/blob/83394493f51991748b9b4706e6d37a8ed874bc05/XeePhotoshopLoader.m, linee 108 ss.

[17]Che indica in dettaglio gli standard che vengono implementati, e le eventuali differenze rispetto

di SQLite.[18] È fondamentale che il *reference manual* sia aggiornato rispetto al codice a cui fa riferimento.

La seconda è la documentazione che interessa agli utilizzatori finali:

- Le istruzioni su come installare e configurare il software;[19]
- Lo *user's manual* (manuale per l'utente) dà istruzioni pratiche su come usare il programma;[20]
- I *known bugs*, un elenco di errori di programmazione conosciuti ma non ancora risolti dagli sviluppatori;[21]
- Il *changelog* (lista dei cambiamenti) o il file *NEWS* sono file che contengono una descrizione dei cambiamenti fra le varie versioni del programma. Per il software scientifico, è importante indicare tutti i cambiamenti che possono influire in maniera significativa sui risultati;[22]
- *Tutorial* e guide, che spiegano in maniera dettagliata come raggiungere un determinato risultato.

Riassumendo, la finalità di questi documenti nel software ad uso scientifico è di garantire che l'utilizzatore finale sia in grado di installare ed utilizzare il programma correttamente, e sia reso consapevole di eventuali *bug* e limitazioni già note agli sviluppatori.

---

agli standard, v. Free Software Foundation, «The GNU C Library Reference Manual, for version 2.38», cit.

[18] Che contiene sia un *reference manual* con diagrammi che spiegano la sintassi del linguaggio SQL, sia numerosi documenti che spiegano come varie funzionalità sono implementate in concreto, e quali considerazioni tecniche hanno portato a quelle scelte; v. https://www.sqlite.org/docs.html.

[19] Questi elementi sono estremamente importanti per garantire il funzionamento corretto e riproducibile del software scientifico.

[20] Laddove il *reference manual* è l'equivalente di un progetto per costruire un prodotto per i produttori, lo *user's manual* è il manuale di istruzioni per i consumatori. È importante che indichi le limitazioni tecniche del programma.

[21] Perché sono bug particolarmente complessi da risolvere, o perché hanno un impatto ridotto. Idealmente, il software non dovrebbe contenere *bug*, ma nella pratica è inevitabile che alcuni *bug* non possano essere risolti immediatamente. Il miglior compromesso è essere messi a conoscenza della presenza di questi *bug*, in modo da tenerli in conto durante la valutazione.

[22] Ad esempio, la risoluzione di un bug, o il cambiamento o eliminazione di un metodo di analisi.

3.1.4 Uso di codice di terze parti

Per "codice di terze parti", si intende il codice scritto da sviluppatori diversi dagli sviluppatori originali o principali.

Nel caso del software scientifico, si potrebbe argomentare che è preferibile evitare il codice di terze parti per due ragioni: non è stato scritto considerando i requisiti particolari dell'informatica forense[23] e non si è familiari con il codice scritto da altri.[24] È possibile confutare entrambe queste nozioni.

I requisiti di funzionamento del software specializzato per l'analisi forense[25] sono condivisi anche dal software "generico". Tutto il software ha interesse a produrre risultati affidabili, l'unica differenza è il bilanciamento fra l'affidabilità e le altre esigenze (efficienza, sviluppo di altre funzionalità, ecc.).[26]

Il codice scritto da altri ha il vantaggio di essere stato sottoposto ad un controllo di qualità diffuso, da parte degli utenti[27] e sviluppatori[28] Specularmente, questi vantaggi diventano svantaggi nel codice nuovo e scritto *ad hoc*. Il fatto che il codice è nuovo significa che è stato messo alla prova solo in un numero limitato di casi, e quindi è intrinsecamente meno affidabile.

Si possono trarre due conclusioni. La prima è l'opportunità di riutilizzare il codice

[23] Ad esempio, nel codice generico è meglio dare una riposta approssimativa in tempi rapidi, ma per l'informatica forense è meglio dare una risposta precisa anche se richiede tempi più lunghi.

[24] Quindi è più difficile stimare se sia affidabile, e modificarlo per allinearlo alle proprie esigenze. Viceversa, con il proprio codice si ha cognizione diretta dei *bug*, dei punti deboli e delle limitazioni, e della sua struttura, quindi può essere modificato in tempi minori.

[25] Lo strumento di analisi deve essere robusto (gli *input* invalidi devono essere rigettati, e gli errori che si verificano devono essere gestiti in maniera adeguata), deve essere preciso (gli *output* sono corretti). L'analisi deve essere ripetibile (non modificare gli *input* originali), riproducibile (gli stessi *input* producono sempre lo stesso *output*), e dettagliata (deve contenere quante più informazioni diagnostiche utili possibile).

[26] È la tensione fra *doing the right thing* (fare la cosa giusta) e *worse-is-better* (il meglio è il nemico del bene), v. sez. "The Rise of Worse is Better" in Richard P. Gabriel, «Lisp: Good News, Bad News, How to Win Big», 2000, https://web.archive.org/web/20070706112430/https://www.dreamsongs.com/Files/LispGoodNewsBadNews.pdf, pp. 7–10.

[27] Quanti più utenti usano il software, ognuno con le proprie configurazioni di hardware e software, tanto più ci si può aspettare che i risultati siano riproducibili, e più *bug* possono essere scoperti e corretti.

[28] Quanti più sviluppatori di terze parti contribuiscono a sviluppare il software, tanto più ci sarà un incentivo a riorganizzare e documentare il codice in maniera che sia di immediata comprensione anche a soggetti che leggono quel codice per la prima volta.

libero già esistente, dove possibile e ragionevole.[29] La seconda è che in teoria, qualsiasi software libero non-specializzato può essere modificato in modo che risponda alle esigenze dell'informatica forense.[30]

Quando si usa codice di terze parti all'interno del software scientifico, è utile usare una tecnica chiamata *vendoring*, ossia, includere una copia integrale del codice di terze parti all'interno del proprio software.[31] Questo permette di evitare vari problemi, come il *dependency hell*,[32] i *supply-chain attacks*,[33] e garantisce la massima riproducibilità del software, perché tutti i componenti necessari sono già inclusi nel progetto.

3.1.5 Controlli di qualità

Esistono vari strumenti che permettono di controllare la qualità del codice, e verificare e garantire che funzioni in maniera corretta e riproducibile anche su sistemi diversi.

I *linter* sono software che controllano se il codice rispetta una serie di regole:[34] possono essere stilistiche,[35] logiche,[36] o riguardare il corretto uso dei tipi nei linguaggi

[29] Ad esempio, perché non esistono ancora soluzioni mature e largamente affermate. Se invece queste soluzioni esistono, è necessario spiegare perché sono inadeguate, ed è preferibile iniziare da zero. In generale, è sempre preferibile evitare di partire da zero. V. J. Spolsky, *Things You Should Never Do, Part I*, 2000, https://web.archive.org/web/20170104073437/https://www.joelonsoftware.com/2000/04/06/things-you-should-never-do-part-i/.

[30] Viceversa, nel caso del software forense proprietario, potrebbe essere impossibile esaminare il codice sorgente per verificare il corretto funzionamento del programma.

[31] V. T. MacWright, 2021, *Vendor by default*, https://web.archive.org/web/20230929010221/https://macwright.com/2021/03/11/vendor-by-default.

[32] Per una definizione del termine e vari esempi, v. O. Barcz, *What is Dependency Hell and How to Avoid it?*, 2021, https://web.archive.org/web/20230208172253/https://www.boldare.com/blog/software-dependency-hell-what-is-it-and-how-to-avoid-it/.

[33] Situazioni in cui un attacco informatico va a compromettere la fonte da cui viene scaricato il software. Ad es., v. D. Goodin, *Backdoor added to PHP source code after breach of internal git server*, 2021, https://web.archive.org/web/20210329192422/https://arstechnica.com/gadgets/2021/03/hackers-backdoor-php-source-code-after-breaching-internal-git-server/.

[34] V. Fondazione OWASP, *Linting Code*, 2022, https://web.archive.org/web/20230328005626/https://owasp.org/www-project-devsecops-guideline/latest/01b-Linting-Code.

[35] Se servono a garantire che il codice sia coerente dal punto di vista estetico. Ad esempio, esistono numerose opzioni ed opinioni su dove posizionare le parentesi graffe, se usare tabulazioni o spazi (e quanti spazi) per l'indentazione, dove inserire i ritorni a capo, la lunghezza massima delle righe di codice, ecc. Queste regole possono essere sempre applicate automaticamente.

[36] Se garantiscono il rispetto delle *best practices* relative a quel linguaggio di programmazione. Ad

dynamically-typed.[37]

I *tests*[38] servono a controllare il corretto funzionamento del programma.[39] Un *test* consiste in un'azione da compiere, ed il risultato previsto. Se il risultato prodotto dall'azione corrisponde al risultato previsto, il *test* è superato (*pass*), altrimenti fallisce (*fail*).

Per il software ad uso scientifico, è estremamente importante controllare la presenza ed il contenuto dei *tests*[40] ed eseguirli[41] prima di usare il programma, per verificare il suo corretto funzionamento. Nella documentazione relativa all'uso del programma è importante indicare come eseguire i *tests* e dare informazioni sommarie riguardo al loro contenuto.[42]

Ancora, è utile adottare la metodologia *TDD* (*test-driven development*, sviluppo guidato dai test), dove i *tests* vengono scritti prima ancora di scrivere il codice. Questo approccio presenta vari vantaggi: permette di scrivere una minore quantità di codice,[43] l'insieme dei *tests* diventa una specificazione formale del funzionamento del

esempio, *shellcheck* contiene centinaia di regole su come scrivere degli *shell script* robusti, v. https://www.shellcheck.net/wiki/. In alcuni casi le correzioni possono essere applicate automaticamente, perché esiste una sola soluzione. Altrimenti, è necessario l'intervento del programmatore.

[37]Ad esempio, v. MyPy, https://mypy-lang.org/.

[38]Esistono varie tipologie di test: gli *unit tests* controllano il funzionamento delle singole "unità" logiche di cui il programma è composto; gli *integration tests* controllano che più unità funzionano correttamente insieme; infine, gli *end-to-end tests* controllano che l'intero programma, dall'avvio fino al termine dell'esecuzione, funziona correttamente. In questa sezione, si userà l'espressione *test(s)* si riferisce a tutte e tre le categorie.

[39]Facendo un'analogia con il diritto, mentre i *linter* svolgono un controllo sugli atti (sul codice), di legittimità (puramente formale), e preventivo (prima dell'esecuzione del programma), i *tests* svolgono un controllo sull'attività (i risultati che vengono raggiunti), che per sua natura è sempre successivo. V. Marcello Clarich, *Manuale di diritto amministrativo*, Società editrice il Mulino, 2022, p. 276–277.

[40]La qualità dei test è molto più importante della loro quantità. Idealmente, i test devono verificare non solo che gli *input* validi siano elaborati correttamente, ma anche, e soprattutto, che gli *input* invalidi vengono correttamente identificati come tali, e rigettati dal programma.

[41]I test non devono essere eseguiti manualmente, uno ad uno. Esistono programmi chiamati *test runner* che individuano i test, li eseguono, e offrono un resoconto dettagliato dei test non superati, in maniera automatica.

[42]Ad esempio, indicare che tipi di test sono inclusi (*unit*, *integration*, e *end-to-end*), come sono strutturati, quante casistiche coprono, se ci sono test che potrebbero fallire su/al di fuori di casi particolari, ma questa situazione non incide sul corretto funzionamento del programma in generale, ecc.

[43]I programmatori sono tenuti a scrivere solo il codice assolutamente necessario per superare i *tests*, invece di creare soluzioni più complesse del necessario. Meno codice è presente all'interno del programma, meno *bug* contiene, e più è facile studiare il suo funzionamento.

programma e dei suoi limiti operativi[44] e i *tests* diventano una forma aggiuntiva di documentazione del codice per gli sviluppatori.[45]

Infine, i test possono essere usati rilevare le *regressions* (regressioni).[46]

I *fuzzers*[47] sono strumenti che verificano l'affidabilità e robustezza del codice fornendo input casuali al programma e osservando quali input causano un *loop* infinito o un *crash*.[48]

Idealmente, il programma deve sempre terminare l'esecuzione in maniera "aggraziata".[49] Nel caso di un *crash*, il programma termina in maniera "brusca", spesso senza dare all'utilizzatore finale indicazioni utili su quale sia il problema.[50]

L'uso dei *fuzzers* permette di rendere il codice più robusto, e capace di gestire il

[44]In altre parole, se tutti i *tests* sono superati, e se i dati forniti al programma al momento dell'utilizzazione pratica rientrano nel tipo di dati che sono gestiti correttamente nei *tests*, la conclusione logica è che il risultato prodotto dal programma sarà corretto. Naturalmente, questa conclusione dipende interamente dalla qualità e quantità dei *tests*.

[45]I *tests* sono una serie di esempi pratici di come usare le funzioni offerte dal codice.

[46]Una regressione è la situazione che si verifica quando un *bug* corretto in precedenza si ripresenta di nuovo, a seguito di cambiamenti nel codice. Per rilevarle in maniera automatica, è possibile aggiungere un *test* che controlla la presenza o meno di quel *bug*.

[47]In inglese, *fuzzy* significa "sfocato" o "confuso", quindi il termine potrebbe essere tradotto in maniera approssimativa come "confonditori".

[48]Se un programma entra in un *loop* infinito non termina mai l'esecuzione. Un *crash* è la situazione in cui l'esecuzione del programma termina in maniera inaspettata, e non prevista dal programmatore, perché una situazione di errore non è stato gestita correttamente. V. Barton P. Miller, Mengxiao Zhang, Elisa R. Heymann, «The Relevance of Classic Fuzz Testing: Have We Solved This One?», *IEEE Transactions on Software Engineering*, vol. 48, fasc. 6, 2022, https://ieeexplore.ieee.org/abstract/document/9309406, pp. 2028–2039, pp. 2030–2031.

[49]Spesso in inglese si usano l'aggettivo *graceful* e l'avverbio *gracefully* per indicare che davanti ad un problema, il sistema non si interrompe in maniera "brusca", ma cerca di continuare l'esecuzione, magari offrendo un risultato solo parziale, se l'errore è *recoverable* (può essere corretto), o interromperla offrendo informazioni diagnostiche utili anche ad utenti non-tecnici, se l'errore è *unrecoverable* (è impossible continuare l'esecuzione). In ogni caso, è importante informare l'utente della presenza di qualsiasi problema che non è abbastanza grave da arrestare l'esecuzione, ma che potrebbe influire sulla qualità e quantità dei dati (questo tipo di problemi vengono indicati come *warnings*, avvertimenti).

[50]I messaggi prodotti a seguito di un *crash* sono indispensabili per i programmatori (affinché possano identificare la loro causa) ma non sono particolarmente utili per gli utilizzatori del software. Il motivo per cui è importante evitare che il programma termini con l'esecuzione con un *crash* e sia sempre in grado di dare una risposta è lo stesso motivo per cui esiste il divieto di *non liquet* per i giudici. Un programma che subisce un *crash* (e quindi né analizza i dati, né indica l'errore che ha impedito l'analisi) è come il giudice che conclude il processo affermando che non ci siano leggi applicabili (e non dà ragione a nessuna delle due parti). Più in generale, la presenza di *crash* può far dubitare della qualità del programma, da un punto di vista anche solo puramente psicologico, prima ancora che tecnico.

numero maggiore di input possibili in maniera "aggraziata".[51]

3.1.6 Riproducibilità e distribuzione del codice

È estremamente importante garantire che il software ad uso scientifico che è stato creato dagli sviluppatori, e che funziona correttamente sulle loro macchine, funzioni nella stessa maniera anche sulle macchine degli utilizzatori finali.[52]

Le tecniche di *reproducible builds*[53] interessano principalmente nell'ambito della sicurezza informatica, perché permettono di rilevare l'inclusione di codice dannoso al momento della compilazione del codice sorgente.[54]

Gli utenti finali non scaricano codice già compilato da terzi,[55] ma scaricano il codice sorgente, lo compilano di persona e verificano che il loro risultato è lo stesso risultato che è stato ottenuto dagli sviluppatori originali.[56] Questo previene l'inclusione di codice dannoso, e più in generale garantisce che il programma si comporterà nello stesso e identico modo sia per gli sviluppatori originali, sia per gli utilizzatori.[57]

I *container* sono un altro strumento utile per garantire la riproducibilità. Un *container* è un ambiente isolato che viene creato all'interno di un sistema operativo già

[51] Inoltre, eseguendo un'analisi statistica dei risultati, diventa possible evidenziare gli errori di programmazione più comuni che diminuiscono l'affidabilità del software, e quindi creare strumenti specializzati per rilevarli. V. B.P. Miller, M. Zhang, E.R. Heymann, *op. cit.*, p. 2033–2036.

[52] Altrimenti, tutti gli sforzi per garantire la qualità del codice fatti fino a questo punto sarebbero inutili, e tutti i vantaggi del software libero diventerebbero lettera morta. Certamente, il codice potrebbe essere studiato, modificato e ridistribuito, ma verrebbe meno la funzione principale, il poter essere eseguito (con risultati riproducibili).

[53] Nel gergo dell'informatica, *build* è il codice macchina che viene prodotto a seguito della compilazione.

[54] In questo tipo di attacco, il codice sorgente è sicuro, ma nel momento in cui viene compilato dall'utente finale per poter essere eseguito, il compilatore aggiunge del codice dannoso. Per l'utente finale, è difficile rilevare la presenza del codice dannoso. Per maggiori dettagli, v. Ken Thompson, «Reflections on trusting trust», *Commun. ACM*, vol. 27, fasc. 8, agosto 1984, https://doi.org/10.1145/358198.358210, pp. 761–763.

[55] I terzi potrebbero modificare il codice sorgente subito prima della compilazione per aggiungere codice dannoso.

[56] Simon Butler et al., «On business adoption and use of reproducible builds for open and closed source software», *Software Quality Journal*, vol. 31, fasc. 3, settembre 2023, https://doi.org/10.1007/s11219-022-09607-z: 687–688.

[57] Questo è il punto che interessa all'informatica forense, la riproducibilità del software di analisi.

in esecuzione, e permette di installare ed eseguire applicazioni.[58] I *container* offrono due vantaggi: sono deterministici[59] e includono soltanto le applicazioni.[60]

Le caratteristiche dei *container* interessano principalmente alla sicurezza informatica,[61] ma risultano comunque utili per l'informatica forense.[62]

3.2 Buone pratiche di sviluppo

3.2.1 Rilevanza per i giuristi

Questa seconda sezione si concentra sulle buone pratiche relative al processo di sviluppo del software, ed indica gli argomenti che possono essere usati per dimostrare che il modello di sviluppo del software libero non è disorganizzato o produce software di qualità inferiore, per il solo fatto che il diritto di modificare e ridistribuire il codice sorgente è garantito a tutti[63] o perché viene solitamente distribuito gratuitamente.[64]

[58] Hanno una funzione analoga alle *virtual machines* (macchine virtuali). La differenza è che le macchine virtuali simulano l'esecuzione di un intero computer e sistema operativo, mentre i *container* servono soltanto a separare le applicazioni all'interno del container dalle applicazioni già presenti sul sistema. V. I. Buchanan, *Containers vs. virtual machines*, 2024, https://web.archive.org/web/20240112012831/https://www.atlassian.com/microservices/cloud-computing/containers-vs-vms.

[59] Ossia, agli stessi input corrispondono gli stessi output. Questo permette di verificare l'integrità del *container*, e di garantire Ad esempio, i container di Docker vengono definiti con un file di testo chiamato "Dockerfile" (v. https://docs.docker.com/develop/develop-images/dockerfile_best-practices/), e sono identificati in maniera univoca da un *hash* (v. sez. "Image digests" in https://docs.docker.com/engine/reference/run/).

[60] I dati sono conservati all'esterno del *container*, in maniera separata ed indipendente. Il *container* che comprende il software di analisi può essere distrutto e ricreato facilmente, in modo da partire sempre dallo stesso ambiente iniziale. Ad esempio, di default Docker conserva i dati generati dall'applicazione solo in maniera temporanea, e quando il *container* viene fermato, i dati vengono persi. V. Docker.org, *Manage data in Docker*, 2024, https://web.archive.org/web/20240106140934/https://docs.docker.com/storage/.

[61] Il fatto che i *container* sono isolati dal resto del sistema serve a limitare i danni derivanti da attacchi informatici, e il fatto che possono essere installati e configurati in maniera automatica elimina eventuali errori umani che potrebbero compromettere la sicurezza del sistema.

[62] La netta separazione fra programmi e dati, e l'installazione e configurazione automatica dei programmi sono utili per garantire la ripetibilità e riproducibilità delle analisi.

[63] Pertanto, si corre il rischio che entrino in circolazione delle versioni modificate in peggio del software originale.

[64] Una massima di esperienza è *ex nihilo nihil fit* (nulla viene dal nulla), e pertanto, se qualcosa non costa nulla, non vale nemmeno nulla.

3.2.2 Progettazione del software

Nel tempo sono state elaborate varie linee-guida per la progettazione del software. Il principio generale che può essere desunto da tutte le altre linee-guida[65] è di scrivere meno software possibile, e di scrivere il software nella maniera più semplice possibile.

È meglio risolvere un problema complesso usando più programmi (relativamente) semplici e generici,[66] che usando un singolo programma complesso e specifico.[67]

Se è assolutamente necessario creare un nuovo programma[68] è utile dividerlo in due parti: il *front-end* gestisce la presentazione dei dati,[69] mentre il *back-end* gestisce la loro trasformazione.[70] Questo permette di estrarre il *back-end* come un componente a sé, chiamato *library* (libreria), in modo che possa essere riutilizzato da altri programmi. Un *framework* raccoglie più librerie, combina le loro funzionalità, e offre ai programmatori un'interfaccia unificata per utilizzarle.[71]

[65] Le regole che vengono menzionate in seguito sono riprese da Eric Steven Raymond, *The Art of Unix Programming*, Addison-Wesley, 2003, http://www.catb.org/esr/writings/taoup/html/, sez. "Basics of the Unix Philosophy".

[66] Dato che i vari programmi devono comunicare fra di loro, è preferibile che lo facciano usando formati liberi e standardizzati. In ogni caso, è utile anche supportare i formati proprietari, ma i formati liberi dovrebbero essere la scelta principale.

[67] Questa è la logica del *glue code*, il codice che "incolla" insieme più programmi, in modo da creare una sorta di "filiera" per i dati. Per automatizzare operazioni meccaniche, che devono essere ripetute più volte, invece di scrivere manualmente tutti i comandi ogni volta, è molto più semplice scrivere del codice che eseguirà i comandi necessari in sequenza.

[68] Perché il problema non può essere risolto combinando insieme più programmi, oppure perché il problema è nuovo, e non esiste ancora un programma in grado di affrontarlo.

[69] *Front-end* significa "parte anteriore", l'interfaccia grafica o testuale con cui l'utente finale interagisce direttamente.

[70] *Back-end* significa "parte posteriore", ed è il "motore" del programma, dove i dati vengono modificati prima di essere mostrati all'utente. Questa impostazione rende più facile modificare, estendere, e verificare il corretto funzionamento del singolo programma, dato che le varie parti sono *loosely-coupled* (accoppiate in maniera non rigida).

[71] Facendo un'analogia con il diritto, mentre le librerie possono essere considerate analoghe alle leggi, perché sono generiche ed astratte, e generalmente serve la mediazione di altri atti per metterle in pratica, i framework possono essere considerati analoghi ai testi unici, perché raccolgono e armonizzano più leggi speciali.

3.2.3 Scelta di una licenza libera

È fondamentale distribuire il software ad uso scientifico con una licenza libera. La Free Software Foundation offre alcune indicazioni per scegliere una licenza.[72]

La Licenza Apache 2.0[73] è non-*copyleft*, e può essere utilizzata per programmi particolarmente semplici[74]

La GNU GPL[75] viene consigliata come la scelta da preferire in generale; La GPL è indicata anche per le librerie software che risolvono problemi nuovi, per cui non esistono altre librerie;[76] Altrimenti, se esistono già altre librerie che svolgono funzioni simili, è preferibile usare la licenza LGPL,[77] che invece non è copyleft.[78]

3.2.4 Sistemi di controllo di versione

I *VCS* (*version control system*, sistemi di controllo di versione) sono lo strumento che più di ogni altro permette lo sviluppo ordinato del software, anche in presenza di più collaboratori.[79]. La funzione di un *VCS* è di tenere traccia dei cambiamenti che

[72]Free Software Foundation, «How to Choose a License for Your Own Work», 2022, https://web.archive.org/web/20220127041134/https://www.gnu.org/licenses/license-recommendations.html.

[73]V. https://web.archive.org/web/20040202124049/http://www.apache.org:80/licenses/LICENSE-2.0.

[74]Ad esempio, il *glue code* che viene utilizzato per l'estrazione o la presentazione di dati. In questo caso, le eventuali modifiche fatte dalla controparte dovrebbero essere comunque comunicate all'interno del contraddittorio.

[75]Dato che esistono più versioni della GPL, è importante aggiungere "e versioni successive", in modo che il software sia sempre compatibile anche con le eventuali versioni successive della GPL, v. Richard Stallman, «For Clarity's Sake, Please Don't Say "Licensed under GNU GPL 2"!», 2022, https://web.archive.org/web/20220219074031/https://www.gnu.org/licenses/identify-licenses-clearly.html.

[76]Questo impone ai programmatori una scelta: scrivere il codice da zero, oppure usare la libreria con licenza GPL, e adottare la licenza GPL anche per il loro programma.

[77]V. https://web.archive.org/web/20070701212426/http://www.gnu.org/licenses/lgpl-3.0.html.

[78]L'idea è di incentivare comunque l'uso del software libero (le altre librerie potrebbero essere non-libere), senza costringere i programmatori ad usare la licenza GPL per il loro programma.

[79]In questa sezione si farà riferimento alla terminologia ed ai concetti usati da Git, un *VCS* sviluppato originariamente per gestire lo sviluppo del kernel Linux, ma attualmente usato dalla stragrande maggioranza degli sviluppatori, e recentemente, persino dalla Microsoft. Oltre a Git, esistono anche altri sistemi. V. R. Donovan, *Beyond Git: The other version control systems developers use*, 2023, https://web.archive.org/web/20230109140009/https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/. Per un'introduzione pratica all'uso di *Git*, si rimanda a Scott Chacon, Ben Straub, «Pro Git. Version 2.1.413», 2023, https://web.archive.org/web/20231223152842/https://github.com/progit/progit2/releases/download/2.1.413/progit.pdf.

sono stati apportati ad uno o più file. Un gruppo di cambiamenti si chiama *commit*,[80] ed i *commits* sono conservati all'interno di un *repository* (deposito). I *VCS* permettono di:

- Ottenere una copia del *repository*;[81]
- Creare nuovi *commits*;[82]
- Sincronizzare i propri *commits* con quelli di altri;[83]
- Lavorare in parallelo su più funzionalità;[84]
- Verificare l'integrità del *repository*;[85]
- Estrarre versioni specifiche del codice dal *repository*;[86]
- Esaminare come un file è variato nel tempo;[87]

[80]Il sostantivo *commit* fa parte del gergo specializzato dell'informatica. In inglese, uno dei possibili significati del verbo *to commit* è "affidare". Pertanto, *commit* può essere tradotto come "affidamento", nel senso che i dati vengono "affidati" al *repository* affinché li conservi.

[81]Il comando *git clone* permette di creare una copia di un intero *repository*. In altre parole, non si ottiene una semplice copia del codice, ma anche della cronologia di sviluppo di quel codice. Git chiama il *repository* originale *origin*.

[82]Il comando *git commit* permette di creare nuovi *commit*. Ogni *commit* è identificato in maniera univoca da un *hash*, che viene calcolato combinando insieme varie informazioni, come i cambiamenti che sono stati apportati ai file rispetto all'ultimo *commit*, l'autore (opzionalmente i commit possono anche essere firmati digitalmente con firma crittografica), la data ed ora di creazione, una spiegazione relativa ai contenuti, ecc. e l'hash dell'ultimo commit. Questo ultimo elemento rende la catena dei commit a prova di manomissione. Se si modifica un commit all'interno del *repository*, si deve ricalcolare il suo hash, e conseguentemente, l'hash di tutti i commit successivi. Questo *repository* manipolato sarà valido, perché gli hash sono corretti, ma conterrà una serie di commit che non esistono con le copie del *repository* non manipolate, e pertanto, non sarà sincronizzabile con esse.

[83]Il comando *git push* invia i commit presenti nella propria copia del *repository* al *repository* originale, mentre *git pull* permette di scaricare i nuovi commit nel *repository* originale nel proprio. È importante notare che *git push* può essere usato solamente dagli utenti che hanno sono stati autorizzati ad usarlo, e non da chiunque.

[84]Il comando *git branch* permette di creare dei "rami". Un "ramo" consiste in una serie di commit, ed un nome per identificarli. Normalmente, un *repository* ha un ramo principale (*master*, *main* o *trunk*), che contiene il codice considerato stabile ed affidabile, e uno o più rami chiamati *topic* o *feature branches*, che contengono i *commit* non ancora considerati stabili. Questo permette agli sviluppatori di sperimentare liberamente, perché è sempre possibile tornare al ramo principale, ed ignorare le modifiche.

[85]Prima di usare il codice, è necessario confermare che il *repository* sia integro. L'uso di *hash* permette di rilevare qualsiasi modifica (accidentale o intenzionale) al *repository* con il comando *git fsck* (abbreviazione di *file-system check*).

[86]Il comando *git checkout* può essere usato per estrarre un commit specifico. È il comando che permette di garantire in piena misura la riproducibilità del software di analisi, anche a distanza di tempo. A questo fine, è necessario indicare l'hash del *commit* è stato usato per svolgere l'analisi nella relazione.

[87]Ad esempio, è possible esaminare un *repository* per studiare i casi di violazione delle licenze. *GlobaLeaks* è distribuito con la licenza *copyleft* AGPL. L'ANAC modifica GlobaLeaks, ed il 14 gennaio

- Esaminare quali *commit* hanno modificato un determinato file;[88]
- Individuare l'esatto *commit* che ha introdotto un certo *bug*.[89]

3.2.5 Contribuzioni di terze parti

Le contribuzioni di terze parti sono una conseguenza naturale dello sviluppo del software libero. Se tutti hanno il diritto di ottenere una copia del codice e modificarlo, chi corregge dei *bug*, o aggiunge delle funzionalità nella propria copia può desiderare di condividere questi miglioramenti con gli sviluppatori originali.

Queste modifiche sono chiamate *patch*.[90] Il solo fatto che un soggetto terzo invia una *patch* agli sviluppatori originali non significa che sarà automaticamente inclusa,[91] perché il software può essere libero e *open-source*, ma *closed-contribution*.[92]

È utile che gli sviluppatori del software libero ad uso scientifico definiscano un processo per accettare le contribuzioni:

- Indicare il tipo di *patch* che gli sviluppatori desiderano;[93]

2019 lo distribuisce sotto il nome di *OpenWhistleblowing* con una licenza diversa, la Licenza Pubblica dell'Unione Europea. Dopo qualche mese, la licenza viene correttamente ripristinata alla AGPL. V. Centro Hermes, *The Italian National Anti-Corruption Authority (ANAC) and the Hermes Center settle a dispute over the aplication of the AGPL license to GlobaLeaks-based OpenWhistleblowing software*, 2020, https://web.archive.org/web/20201019132745/https://www.globaleaks.org/anac-and-the-hermes-center-settle-a-dispute-over-the-application-of-the-agpl-license-to-globaleaks-based-openwhistleblowing-software/. È possibile visualizzare i *commits* che hanno modificato il file "LICENSE" (v. https://github.com/anticorruzione/openwhistleblowing/commits/master/LICENSE), e notare che la violazione della licenza AGPL è iniziata il 14 gennaio 2019, ed è terminata il 24 ottobre 2019.

[88] Il comando *git blame* (letteralmente, "dare la colpa"), permette di associare ad ogni riga di codice l'ultimo *commit* che l'ha aggiunta o modificata.

[89] Usando il comando *git bisect*.

[90] Letteralmente, "pezze", perché saranno metaforicamente "cucite" all'interno del codice.

[91] Questo è forse il malinteso più comune riguardo il software libero: se il codice è libero, e quindi chiunque è libero di inviare *patch*, allora qualsiasi contribuzione sarà accettata.

[92] Ad esempio, *litestream* era un progetto che non accettava contribuzioni da parte di terzi. V. B. Johnson, *litestream*, 2021, https://github.com/benbjohnson/litestream/tree/4d41652c12c182d7f0721cc8eda0e3c78d98bae0. Attualmente il progetto accetta contribuzioni di terze parti.

[93] È utile che gli sviluppatori indichino sia i *goals* (obiettivi) del progetto, che i *non-goals* (obiettivi al di fuori dell'ambito del progetto). I *non-goals* permettono di concentrare gli sforzi di sviluppo sugli elementi realmente essenziali, e di evitare il *feature creep* (eccesso di funzionalità). È preferibile avere pochi strumenti di analisi altamente affidabili, che un grande numero di metodi il cui funzionamento non è stato verificato in maniera esaustiva.

- Definire i requisiti per la *patch*,[94] e le modalità di *code review* (revisione del codice), per verificare la qualità del codice;[95]
- Definire i soggetti che hanno il *write access* (accesso in scrittura) sul *repository* disponibile al pubblico, e che contiene il codice sorgente;[96]
- Infine, è utile richiedere l'accettazione di un *CLA* (*contributor license agreement*, accordo sulla licenza per chi contribuisce) come condizione per includere il codice di terze parti.[97]

La recente *backdoor*[98] trovata all'interno del software *xz* (CVE-2024-3094) non dimostra che il modello di sviluppo aperto può facilitare gli attacchi informatici[99]

[94]Questi requisiti interessano ai contributori esterni. Ad esempio, il kernel Linux ha delle linee-guida dettagliate sul contenuto e formato per le patch. V. Comunità di sviluppo del kernel, *Submitting patches: the essential guide to getting your code into the kernel*, n.d., https://web.archive.org/web/20180828081227/https://www.kernel.org/doc/html/v4.17/process/submitting-patches.html

[95]Questi elementi riguardano l'organizzazione interna del progetto, ma per motivi di trasparenza, è preferibile renderli pubblici. In particolare, si devono indicare la persona o persone che andranno a controllare che la *patch* sia pertinente al progetto, che segue lo stile del codice seguito dal progetto, che non presenti bug manifestamente evidenti, e che non contenga contribuzioni in mala fede. Ad esempio, c'è stato un esperimento in cui dei ricercatori hanno inviato delle *patch* contenenti bug al kernel Linux, allo scopo di verificare se gli addetti alla *code review* le avrebbero rigettate o meno. V. Quishi Wu, Kangjie Lu, «On the Feasibility of Stealthily Introducing Vulnerabilities in Open-Source Software via Hypocrite Commits», 2021, https://web.archive.org/web/20210928192905/http://www.coding-guidelines.com/code-data/OpenSourceInsecurity.pdf e S. J. Vaughan-Nichols, *Greg Kroah-Hartman bans University of Minnesota from Linux development for deliberately buggy patches*, 2021, https://web.archive.org/web/20210421203003/https://www.zdnet.com/article/greg-kroah-hartman-bans-university-of-minnesota-from-linux-development-for-deliberately-buggy-patches/.

[96]Questo è il punto più importante. Anche se chiunque può scaricare una copia del *repository*, modificarla, e inviare una *patch*, soltanto alcune persone possono effettivamente includere la patch all'interno del *repository* originale. Volendo, è possibile creare un *fork* (bivio), ossia, una copia del *repository* originale che viene gestita da sviluppatori diversi, e contiene le modifiche di questi ultimi. Di solito i *fork* sono creati se il progetto originale è stato abbandonato, oppure se gli sviluppatori originali si rifiutano di includere delle *patch*.

[97]L'obiettivo è di evitare dubbi riguardo a chi appartengano i diritti intellettuali relativi al codice. Ad esempio, la FSF richiede che i terzi, che contribuiscono codice a programmi di cui la FSF detiene il diritto d'autore, conferiscano alla FSF i diritti d'autore relativi alla loro contribuzione. V. E. Moglen, *Why the FSF Gets Copyright Assignments from Contributors*, 2022, https://web.archive.org/web/20220102214048/https://www.gnu.org/licenses/why-assign.html.

[98]Un tipo di attacco informatico, dove si inserisce una "porta di servizio" nel codice, che permette l'esecuzione di comandi arbitrari.

[99]Al contrario, è stato proprio il fatto che il codice sorgente era disponibile che ha permesso ad un ricercatore di sicurezza di studiare il funzionamento della *backdoor*. V. A. Freund, *backdoor in upstream xz/liblzma leading to ssh server compromise*, 2024, https://web.archive.org/web/20240403232949/https://openwall.com/lists/oss-security/2024/03/29/4.

ma al contrario, dimostra la necessità di svolgere una rigorosa *code-review* anche nei confronti di *patch* introdotte da sviluppatori considerati affidabili.[100]

3.2.6 Sviluppo trasparente del software

In generale, lo sviluppo del software libero deve essere condotto nella maniera più trasparente possibile.[101] Chi sviluppa software libero è tenuto (idealmente) a condividere non solo il codice, ma anche:

- Informazioni come la documentazione progettuale, le discussioni fra gli sviluppatori che sono rilevanti per il progetto, gli obiettivi di medio e lungo termine, e così via;[102]
- Le qualificazioni dei soggetti che hanno contribuito al progetto;[103]
- Le discussioni fra utenti, o fra sviluppatori ed utenti, su come usare il software;[104]
- Le discussioni riguardo la segnalazione di *bug*;[105]
- Le discussioni a seguito dell'invio di *patch*;[106]

[100]V. sez. "Backstory" in Akamai Security Intelligence Group, *XZ Utils Backdoor — Everything You Need to Know, and What You Can Do*, 2024, https://web.archive.org/web/20240403225340/https://www.akamai.com/blog/security-research/critical-linux-backdoor-xz-utils-discovered-what-to-know.

[101]Per "trasparenza" si fa riferimento allo stesso valore tipico del diritto amministrativo, che serve a garantire un controllo diffuso sull'attività ed organizzazione nel diritto amministrativo. V. M. Clarich, *op. cit.*, pp. 153–154, 306–307.

[102]Sono tutte informazioni che hanno rilevanza solo "interna", ma che comunque non sono "riservate", e che quindi è opportuno rendere disponibili al pubblico, anche se l'utilità di condividerle non è immediatamente apparente. Anche il solo fattore psicologico di sapere che il progetto si comporta come se "non abbia nulla da nascondere" ispira fiducia nell'opera degli sviluppatori.

[103]Nel diritto amministrativo, è fondamentale dimostrare che le persone abbiano le competenze necessarie per ricoprire una determinata carica. Nello sviluppo del software libero, è utile dimostrare che il software non viene necessariamente sviluppato solo da dilettanti e volontari, ma anche da persone con esperienza professionale, o impiegati di compagnie che hanno pubblicato del software libero.

[104]Si potrebbe affermare che queste discussioni sono una forma ulteriore di documentazione per il software, specie se rispondono a domande come, "Come posso fare per …?" (spiegano come raggiungere un certo risultato), oppure "È possibile usare questo software per …?" (definiscono la destinazione d'uso del software).

[105]Anche queste discussioni sono una forma di documentazione, perché individuano le situazioni in cui un bug si verifica, le possibili soluzioni alternative da usare fino alla sua soluzione, indicano come il bug è stato risolto, possono essere utili per la soluzione di bug futuri, ecc.

[106]Così come le discussioni per i *bug*, formano una sorta di "documentazione storica" per il codice, e servono a spiegare le motivazioni per cui il codice è stato modificato in un certo modo.

- Impostare un sistema di *CI* (*continuous integration*, integrazione continua).[107]

Per gestire quanto sopra indicato, si possono seguire due strade.

La prima consiste nell'usare servizi integrati, ma proprietari, come GitHub[108] o GitLab.[109] Il vantaggio è la facilità d'uso,[110] lo svantaggio è la dipendenza da una piattaforma proprietaria.[111]

La seconda è di usare alternative ai servizi proprietari che siano software libero, come Gitea,[112] e fare *self-hosting*.[113] Il vantaggio è che si ottiene il pieno controllo della piattaforma,[114] ma al tempo stesso, si diventa responsabili dell'amministrazione del server su cui i dati risiedono.[115]

[107]Un sistema CI permette di eseguire i *tests* e compilare il software in maniera automatica, appena un *commit* viene pubblicato, o prima di integrare una *patch* all'interno del codice. Possono essere impostati per verificare che il software funzioni correttamente su più piattaforme e configurazioni, e forniscono informazioni dettagliate a seguito di problemi. L'uso di un sistema di CI permette di individuare i bug il prima possibile, e garantisce che il software continua a funzionare correttamente nel corso dello sviluppo. V. RedHat.com, *What is CI/CD?*, https://web.archive.org/web/20231213065115/https://www.redhat.com/en/topics/devops/what-is-ci-cd.

[108]V. https://github.com/.

[109]V. https://about.gitlab.com/.

[110]I servizi offrono interfacce grafiche per molte operazioni, sono ampiamente documentati, i vari componenti (gestione del codice, delle discussioni e bug, CI) sono già installati e configurati.

[111]Non si ha il pieno controllo della piattaforma (ad esempio, GitHub aveva temporaneamente disattivato il repository di *youtube-dl* a seguito di una diffida da parte della RIAA: v. https://github.com/github/dmca/blob/1de32ff91eba5b48334b04d72bc69aa6cbb50359/2020/10/2020-10-23-RIAA.md; il repository fu riattivato dopo alcune settimane, senza modifiche significative al codice: v. A. Maxwell, *GitHub Reinstated YouTube-DL But Restoring Forks is Apparently a Problem*, 2021, https://web.archive.org/web/20210417214135/https://torrentfreak.com/github-reinstated-youtube-dl-but-restoring-forks-is-apparently-a-problem-210417/), si può cadere in situazioni di *vendor lock-in* (si rimane "intrappolati" su un servizio, perché non offre il modo di esportare i dati), il servizio può usare il codice o dati per finalità commerciali e senza il permesso degli sviluppatori (ad esempio, si pone il problema se il codice rilasciato con licenza GPL possa essere usato per creare intelligenze artificiali che generano codice: v. FOSSA Editorial Team, *Analyzing the Legal Implications of GitHub Copilot*, 2021, https://web.archive.org/web/20221105223552/https://fossa.com/blog/analyzing-legal-implications-github-copilot/).

[112]V. https://github.com/go-gitea/gitea, che ha un'interfaccia molto simile a quella delle piattaforme proprietarie.

[113]Un servizio *self-hosted* è un servizio che viene amministrato *in-house*, dalla stessa persona che lo usa, e non da terzi.

[114]Gitea è software libero, quindi è possibile leggere e modificare il codice sorgente, e non ci si deve preoccupare il gestore del servizio possa sospendere l'accesso al repository.

[115]Con i servizi proprietari, lo sviluppatore non si deve minimamente preoccupare della gestione del server (installazione e configurazione del software, configurazione del server, installazione degli aggiornamenti, garanzia dell'*uptime* e della sicurezza informatica, ecc.) su cui risiedono i dati. Viceversa, con il *self-hosting*, il rischio di attacchi informatici contro il codice aumenta notevolmente, dato che gli

sviluppatori individuali non hanno le stesse risorse e competenze di una grande compagnia.

Capitolo 4

Software libero per l'informatica forense

4.1 Uso del software libero nella pratica

I vantaggi del software libero descritti fino a questo momento non sono destinati a rimanere lettera morta, perché esistono già programmi che possono essere utilizzati nell'ambito delle attività dell'informatica forense. Questo capitolo darà un resoconto non esaustivo[1] del software libero esistente, inclusi i sistemi operativi liberi.

Si potrebbe dire che l'informatica forense è una disciplina che studia e gestisce una serie di *black boxes*:[2] l'hardware e software oggetto di analisi, e l'hardware e software con cui viene condotta l'analisi, le reti e protocolli con cui le macchine comunicano fra di loro, e così via.

Allo stesso tempo, si è dimostrato come l'informatica forense deve soddisfare

[1] Cercare di individuare tutto il software esistente per ciascuna branca dell'informatica forense andrebbe fuori dall'ambito della trattazione. In ogni caso, il software è in continua evoluzione, e le considerazioni svolte in questo momento potrebbero non valere nel futuro. Pertanto, ci si limiterà a svolgere considerazioni generiche.

[2] Nel gergo dell'informatica, una *black box* (scatola nera) è uno strumento di cui si può osservare il comportamento esterno, ma non si conosce l'esatto meccanismo di funzionamento interno. Non si sta facendo riferimento alle "scatole nere" che vengono recuperate a seguito di incidenti di aeromobili o imbarcazioni (che comunque possono essere di interesse per l'informatica forense, se contengono dati digitali).

le esigenze del contraddittorio nel processo, e quindi si deve essere in grado di spiegare nella maniera più completa e dettagliata possibile come si è giunti ad una certa conclusione. Per questo motivo, si è sostenuta l'utilità del software libero per le operazioni di analisi.

4.2 Sistema operativo libero

Il sistema operativo è il software[3] più "fondamentale". Come qualsiasi altro programma, deve essere installato ed eseguito,[4] ma la sua funzione è di fornire tutti gli elementi necessari per il funzionamento di altri programmi.[5]

Ancora, così come esiste software libero, esistono anche sistemi operativi (quasi) interamente liberi.[6] Un sistema operativo libero[7] presenta tutti i vantaggi tipici del software libero. In particolare, è possibile studiare il funzionamento del sistema,[8] ed è

[3]Meglio, "collezione di software", dato che i sistemi operativi moderni sono complessi, e composti da numerosi programmi. Ad esempio, il progetto "Linux From Scratch" offre una guida su come creare un sistema Linux partendo da zero, e compilando tutto il software necessario usando il codice sorgente. V. Gerard Beekmans, *Linux From Scratch. Version 12.0*, Bruce Dubbs (a cura di), 2023, https://web.archive.org/web/20230901165736/https://www.linuxfromscratch.org/lfs/view/stable/.

[4]Spesso il sistema operativo è già installato sui computer, e al primo avvio deve soltanto essere configurato per la prima volta. In ogni caso, il sistema operativo è il primo programma che viene eseguito quando il computer viene avviato.

[5]Ad esempio, il sistema operativo rileva e gestisce tutti i componenti hardware connessi al computer (inclusi i supporti di memoria), gestisce la memoria del computer e l'esecuzione dei vari programmi, ecc.

[6]In alcuni casi, è necessario includere software non-libero per far funzionare alcuni componenti hardware, come la connessione Wi-Fi, o la scheda video. Ad esempio, il programma per installare la distribuzione GNU/Linux Debian tradizionalmente non includeva questo tipo di software, perché per motivi ideologici, voleva rimanere un sistema composto interamente da software libero. L'installazione di questo software doveva essere fatta manualmente, in modo che l'utente sia pienamente consapevole che il sistema contiene componenti non-liberi. Tuttavia, seguito di una discussione nel progetto il programma per l'installazione è stato modificato, ed il programma per l'installazione di Debian 12 adesso include anche i componenti non-liberi. V. Autori di Debian Wiki, *Firmware*, 2023, https://web.archive.org/web/20230720195706/https://wiki.debian.org/Firmware.

[7]Fatta salva l'eccezione dei componenti non-liberi, il cui impatto sul funzionamento del sistema è limitato (sono necessari per usare alcuni componenti hardware, ma non pregiudicano il corretto funzionamento del software).

[8]Dato che si ha accesso al codice sorgente è possibile verificare come il sistema operativo è stato progettato, e valutare la presenza di elementi che possono influire in positivo o negativo sull'affidabilità delle analisi. Dal punto di vista processuale il margine di discussione all'interno del contraddittorio è molto più ampio rispetto al software proprietario. Si riduce così il numero di *black boxes* all'interno del procedimento, perché almeno gli strumenti di analisi sono liberamente esaminabili.

possibile creare e distribuire copie del sistema.[9]

I sistemi operativi liberi con la maggiore utilizzazione sono le distribuzioni GNU/Linux.[10] Generalmente, ogni distribuzione offre un proprio *package manager* (gestore di pacchetti).[11] I pacchetti contengono di solito contengono software,[12] ma possono contenere anche soltanto dati.[13] I pacchetti sono generalmente preparati dagli stessi sviluppatori della distribuzione,[14] ma è possibile per sviluppatori terzi creare i propri pacchetti.[15] Distribuzioni diverse usano strategie diverse, ciascuna con i propri vantaggi e svantaggi:

- Le distribuzioni *fixed-point* sono stabili e ben testate,[16] ma contengono software datato;[17]
- Le distribuzioni *rolling*[18] offrono il software più recente, al costo di una minore

[9] È possibile creare una copia dell'intero ambiente di analisi (inteso come sistema operativo e programmi installati) che è stato usato da una parte processuale, offrirlo alla controparte, e conservarlo nel caso in cui sia necessario ripetere le analisi in un momento successivo.

[10] "GNU" è il nome del sistema operativo (il software di base necessario per il funzionamento del sistema, come un programma per eseguire comandi, editor di testo, compilatori, programmi per visualizzare i manuali, ecc.), "Linux" è il nome del *kernel* (il componente del sistema operativo che gestisce l'hardware, come supporti di memoria, schede audio, video e di rete, la RAM, tastiera, mouse, altre periferiche, ecc.). Spesso si usa solo il termine "Linux" per riferirsi in maniera generica alle distribuzioni Linux, ma è improprio. Non esiste "un" sistema operativo chiamato Linux, e "Linux" di per sé non è un sistema operativo, ma uno dei componenti necessari per un sistema operativo. Per dettagli, v. Richard Stallman, «Linux and the GNU System», 2021, https://web.archive.org/web/20211109122924/http://www.gnu.org/gnu/linux-and-gnu.en.html.

[11] Ad esempio, Debian e Ubuntu usano APT, Fedora usa DNF, Arch Linux usa Pacman, ecc.

[12] Di solito come codice macchina, ma è possibile scaricare pacchetti che contengono solo codice sorgente. I pacchetti includono anche i file di configurazione, la documentazione del software, ecc.

[13] I pacchetti possono contenere risorse aggiuntive per il software. Ad esempio, font, dizionari aggiuntivi per un correttore ortografico, ecc.

[14] Alcune distribuzioni offrono soltanto ed esclusivamente software libero (per una lista, v. https://www.gnu.org/distros/free-distros.en.html), mentre altre permettono di installare anche software proprietario, se l'utente lo desidera.

[15] Ad esempio, il software proprietario viene spesso reso disponibile per Linux come un pacchetto in formato DEB e/o RPM, che possono essere installati rispettivamente su Ubuntu e Fedora con APT e DNF.

[16] Dato che i rilasci sono infrequenti, è possibile garantire che il software contenga meno bug possibili, e fra i rilasci, si può contare sul fatto che installare il software è un'azione riproducibile, perché non ci sono cambiamenti.

[17] Distribuzioni che aggiornano i pacchetti in maniera "sincrona": aggiornare la distribuzione permette anche di aggiornare i pacchetti, ma fino al rilascio della nuova versione della distribuzione (che potrebbe avvenire dopo mesi o anni), i pacchetti ricevono solo aggiornamenti relativi per la sicurezza e bug particolarmente gravi.

[18] Distribuzioni che aggiornano i pacchetti in maniera "asincrona": ogni pacchetto può essere

stabilità e riproducibilità;[19]

- Nix è un *package manager* che è in grado di installare più versioni dello stesso programma senza che entrino in conflitto.[20] NixOS è una distribuzione Linux che usa Nix come il suo *package manager*.[21]

Con tutti i *package manager* si pongono due problemi: l'affidabilità di dati scaricati da internet[22] (risolto con tecniche come l'uso di firme digitali)[23] e l'affidabilità del software compilato da altri[24] (risolto con tecniche di *reproducible build*).

Nel tempo, data la flessibilità offerta dalle distribuzioni GNU/Linux, sono state create delle distribuzioni specializzate per l'informatica forense, che raccolgono software libero, tra cui CAINE,[25] DEFT,[26] SIFT Workstation,[27] Kali Linux,[28] BackBox Linux,[29] ed altre. Le distribuzioni specializzate hanno varie caratteristiche in comune:

- Possono essere avviate in modalità *live*,[30] oltre che essere installate su un

aggiornato appena viene rilasciata una nuova versione.

[19]Data la frequenza degli aggiornamenti, non è possibile verificare il corretto funzionamento di ogni pacchetto, e installare o aggiornare il sistema in momenti diversi produce risultati diversi, perché non è possibile prevedere in anticipo quali pacchetti saranno installati.

[20]La descrizione di Nix è oggetto di una tesi di dottorato, v. Eelco Dolstra, «The purely functional software deployment model», Utrecht University, 2006, https://dspace.library.uu.nl/handle/1874/7540.

[21]Eelco Dolstra, Andres Löh, «NixOS: a purely functional Linux distribution», 2008, https://github.com/edolstra/edolstra.github.io/blob/2eed3fdbff656d01fe5372e9bf322799de0eaba7/pubs/nixos-icfp2008-submitted.pdf, p. 1.

[22]Per ragioni di efficienza, i pacchetti vengono offerti da più *mirrors* (lett. "specchi"), ossia server che offrono una copia dei contenuti già disponibili su un altro server. Tuttavia, ogni *mirror* può modificare i dati, dopo che ha ottenuto una copia. Ancora, mentre i dati sono in transito, è possibile che possano essere alterati, ad esempio, con tecniche di *deep packet inspection*.

[23]Chi prepara il pacchetto vi applica anche la sua firma digitale prima di distribuirlo, per garantire la sua integrità ed autenticità. Successivamente, il gestore di pacchetti verifica che la firma digitale sia valida, e in presenza di errori non installa i pacchetti.

[24]In molti casi gli sviluppatori della distribuzione devono modificare il codice sorgente dei programmi per adattarlo alle peculiarità della distribuzione. È fondamentale che queste modifiche siano messe a disposizione del pubblico. Tuttavia, in generale, gli sviluppatori potrebbero introdurre qualsiasi modifica, e sarebbe difficile rilevarle da parte degli utilizzatori della distribuzione.

[25]V. https://web.archive.org/web/20240119162336/https://www.caine-live.net/.

[26]Non più in sviluppo. Per un archivio della pagina principale del progetto, v. https://web.archive.org/web/20190101021304/http://www.deftlinux.net/.

[27]V. https://web.archive.org/web/20240111183952/https://www.sans.org/tools/sift-workstation/.

[28]V. https://web.archive.org/web/20240305095732/https://www.kali.org/.

[29]V. https://web.archive.org/web/20240328124301/https://linux.backbox.org/.

[30]Ossia, il sistema operativo viene copiato nella RAM, senza essere installato sul computer in maniera

computer;[31]

- Prendono ogni precauzione per evitare operazioni in scrittura sui dispositivi che vengono collegati;[32]
- Includono del software pre-installato, in modo da garantire la piena riproducibilità dell'ambiente di analisi, e quindi la ripetibilità delle analisi;[33]
- Offrono delle interfacce grafiche e strumenti software *ad-hoc* per velocizzare le operazioni tipiche e la generazione del report finale.[34]

4.3 Software libero per acquisire i dati

Il prerequisito per analizzare i dati è la loro corretta acquisizione. L'acquisizione è generalmente un atto irripetibile[35] e pertanto è assolutamente necessario garantire la massima trasparenza e affidabilità dell'operazione usando software libero.[36]

Se i dati risiedono su un supporto materiale che può essere collegato ad un computer[37] è possibile usare *dd* o *ddrescue*.

ddrescue è un comando specializzato per copiare dati da supporti che possono presentare errori di lettura. Usa un algoritmo creato *ad hoc* per cercare di copiare quanti più dati possibile, causando meno danni possibile al supporto.[38] Inoltre, produce anche un file di log chiamato *mapfile*, che permette di interrompere e riprendere l'operazione

fissa. Questo permette di usare la distribuzione direttamente sul sistema oggetto di acquisizione (ad esempio, se non è possibile rimuovere i supporti contenuti al suo interno), e garantisce la massima riproducibilità dell'ambiente di analisi, e ripetibilità dell'analisi (perché qualsiasi modifica fatta al sistema viene persa dopo che il computer viene riavviato).

[31] E. Huebner, S. Zanero, *op. cit.*, p. 75.

[32] *Ibidem*, p. 73.

[33] *Ibidem*, pp. 76–78.

[34] *Ibidem*, pp. 74–76, 78–79.

[35] Esiste il rischio che il supporto materiale subisca modifiche durante l'acquisizione, ed il rischio di irripetibilità sopravvenuta, perché i dati si danneggiano o vengono cancellati.

[36] L'accesso al codice sorgente e la possibilità di distribuire liberamente copie garantiscono la trasparenza, e permettono di valutare in maniera consapevole se il software sia affidabile o meno. Viceversa, nel caso di software proprietario ci si deve affidare ciecamente al prodotto che viene fornito, dato che non è possibile sapere in maniera altrettanto trasparente come è stato sviluppato.

[37] Ad esempio, un *hard disk* interno o esterno, memorie flash USB o SD, supporti ottici, ecc.

[38] V. sez. 4, "Algorithm" in Antonio Diaz Diaz, «GNU ddrescue Manual», 2023, https://web.archive.org/web/20240109210952/https://www.gnu.org/software/ddrescue/manual/ddrescue_manual.html.

di acquisizione, e contiene informazioni diagnostiche dettagliate sullo stato di ogni settore letto dal disco.[39]

Se non è possibile usare *ddrescue*, si può usare *GNU dd*.[40] Il vantaggio principale di *dd* è la sua ubiquità sui sistemi GNU/Linux, dato che è un comando standard.[41] Lo svantaggio principale è la sua semplicità: *dd* è un comando generico, e non offre meccanismi sofisticati di gestione degli errori,[42] o informazioni diagnostiche dettagliate.[43]

Se i dati sono su *embedded devices*[44] è necessario controllare le opzioni disponibili, caso per caso. In generale, si possono seguire due strade.

Potrebbe essere possibile svolgere l'acquisizione usando lo stesso dispositivo, se (congiuntamente) quest'ultimo usa un sistema operativo basato su GNU/Linux, è possibile connettersi da remoto a quel dispositivo, è possibile eseguire comandi sul dispositivo ed è possibile trasmettere dati all'esterno del dispositivo.[45]

In alternativa, nel caso in cui il produttore del dispositivo offre uno strumento *ad hoc* per l'estrazione di dati o creazione di backup, è possibile eseguirlo all'interno di una macchina virtuale creata con software libero.[46]

---

[39] V. sez. 8, "Mapfile structure", in *ivi*.

[40] V. sez. 11.2, "dd: Convert and copy a file" in Free Software Foundation, «GNU Coreutils», cit.

[41] Fa parte delle *GNU coreutils*, e la sua presenza è richiesta dallo standard *Linux Standard Base*.

[42] Di default, *dd* si arresta dopo il primo errore di lettura. È possibile usare le opzioni *conv=noerror,sync* affinché *dd* continui a seguito di errori, e riempia le parti che non è stato possibile leggere con zeri.

[43] *dd* non produce un *mapfile*, e quindi offre molte meno informazioni rispetto a *ddrescue*.

[44] Intesi come dispositivi per cui non è possibile estrarre il supporto di memoria che contiene i dati. Ad esempio, gli *smartphones*, apparecchiature mediche, autoveicoli, ecc.

[45] Ad esempio, se si può accedere al dispositivo mediante *ssh*, si può usare *md5sum* e *sha1sum* per calcolare l'hash dei file, usare *tar* per creare un archivio con i file da estrarre, *scp* per copiarlo all'esterno. Naturalmente, nella pratica è necessario verificare la presenza di comandi utili per l'operazione di acquisizione.

[46] Ad esempio, con *VirtualBox* o *QEMU*. È preferibile usare una macchina virtuale in modo da tenere traccia dell'ambiente che è stato usato per l'acquisizione dei dati. Il programma per l'estrazione potrebbe essere anche proprietario, ma non è un problema, perché si può presupporre che uno strumento per la creazione di backup creato dal produttore stesso del dispositivo abbia tutto l'interesse a copiare i dati nella maniera più completa ed affidabile possibile. Esistono anche strumenti per la *mobile forensics* di terze parti in grado di acquisire i dati, ma si pone il problema della loro affidabilità. Se lo strumento è in grado di leggere tutti i dati presenti sul dispositivo, ma deve forzare le misure di sicurezza esistenti per farlo, e non può spiegare in dettaglio come le ha forzate (per ragioni di segreto industriale, perché altrimenti il produttore del dispositivo rilascerebbe un aggiornamento di sicurezza, ecc.), è difficile potersi fidare a pieno della genuinità del risultato.

Purtroppo, entrambe le modalità presentano tre svantaggi, difficilmente superabili: la dipendenza da strumenti proprietari,[47] la quantità di dati che è possibile estrarre,[48] e la loro irripetibilità intrinseca.[49]

Per la *network forensics*, Wireshark[50] è un software maturo, in sviluppo da più di 20 anni, e permette di acquisire ed analizzare tutto il traffico di rete. In particolare, con alcuni accorgimenti, può essere usato per eseguire l'acquisizione forense di pagine web.[51]

Esistono anche programmi che, pur non essendo stati sviluppati specificamente per l'informatica forense, permettono di acquisire quante più informazioni possibili da servizi proprietari disponibili su internet.[52] Ad esempio:

- *Rclone*[53] permette di acquisire dati da piattaforme cloud;[54]
- *Yt-dlp*[55] può scaricare video da Youtube e numerose altre piattaforme;[56]
- *Instaloader*[57] può scaricare immagini e video da Instagram;
- *DiscordChatExporter*[58] permette di estrarre un log dei messaggi su Discord.

[47] In particolare, non si ha accesso al codice sorgente del sistema operativo del dispositivo oggetto di acquisizione, o dello strumento con cui si crea il backup del dispositivo.

[48] È molto probabile che esistano meccanismi di sicurezza che impediscono l'accesso a tutti i file.

[49] Le operazioni di acquisizione vengono svolte mentre il dispositivo è acceso, e l'esecuzione dei comandi o del backup modifica lo stato del dispositivo.

[50] V. https://web.archive.org/web/20240101043715/https://www.wireshark.org/.

[51] In primo luogo, è necessario impostare Wireshark in modo che possa catturare il traffico generato dal *browser* (v. sez. "Using the (Pre)-Master-Secret" in https://web.archive.org/web/20230724183942/https://wiki.wireshark.org/TLS). Dopo che Wireshark ha iniziato ad acquisire i pacchetti, ma prima e dopo che le pagine da acquisire sono state visitate, è utile visitare il sito internet di una testata giornalistica, come modo per provare che la cattura è avvenuta in un determinato momento. Ancora, è utile registrare lo schermo durante lo svolgimento delle operazioni, ed è utile eseguirle all'interno di una macchina virtuale, in modo da lasciare quante più tracce possibili dello svolgimento dell'operazione.

[52] In molti casi, è possibile acquisire questi contenuti usando un semplice web browser. Tuttavia, per acquisire i contenuti in blocco, è preferibile usare uno strumento specifico. Nell'usare questi strumenti, è importante controllare la documentazione, e attivare le opzioni che forniscono all'utente quante più informazioni diagnostiche possibile.

[53] V. https://web.archive.org/web/20240103231619/https://rclone.org/.

[54] Ad esempio, Dropbox, Google Drive, OneDrive, ecc. Per una lista completa, v. https://web.archive.org/web/20240112155053/https://rclone.org/overview/.

[55] V. https://github.com/yt-dlp/yt-dlp.

[56] V. https://github.com/yt-dlp/yt-dlp/blob/8463fb510a58050ec118b3ae17bf00d08ea7b881/supportedsites.md.

[57] V. https://github.com/instaloader/instaloader.

[58] V. https://github.com/Tyrrrz/DiscordChatExporter.

FIT[59] è un software integrato per l'acquisizione forense di pagine web, che combina le funzionalità di un gran numero di librerie e programmi di terze parti[60] con gli strumenti utili per l'informatica forense.[61] Il fatto che sia un software relativamente recente[62] non deve scoraggiare il suo uso, perché allo stesso tempo, è anche un software concettualmente semplice,[63] e pertanto, la quantità di codice realmente "innovativa", di cui si deve provare il corretto funzionamento, è limitata.

In alcuni casi, i servizi offrono già strumenti per esportare dati,[64] ma questi servizi sono proprietari.[65] Tuttavia, in alcuni casi lo strumento per esportare dati potrebbe essere software libero. Ad esempio, il client di Telegram è software libero[66] e permette di scaricare una copia dei dati relativi all'account di un utente.[67]

La *memory forensics*[68] tradizionalmente veniva quasi completamente ignorata[69] perché presentava (e continua a presentare) varie difficoltà. Le memorie volatili si disperdono rapidamente appena il dispositivo viene spento,[70] ma se il dispositivo è acceso, l'acquisizione della memoria va a modificare lo stato della memoria stessa.[71]

[59] Acronimo di *Freezing Internet Tool*, v. https://github.com/fit-project/fit.

[60] Tra cui anche i già menzionati *yt-dlp* e *instaloader*, perché sono inclusi nella lista delle dipendenze. V. https://github.com/fit-project/fit/blob/cbf8d43d8dc82587b93fbfdb9e0dc14ec0b94ef1/poetry.lock.

[61] Come la cattura del traffico di rete, la registrazione dello schermo, il calcolo di più *hash* per i file scaricati, la generazione automatica di un resoconto delle operazioni compiute.

[62] FIT è stato pubblicato per la prima volta nel 2021 da Fabio Zito (v. https://github.com/zitelog/fit), come tentativo di riprodurre le funzionalità del software proprietario FAW (v. https://web.archive.org/web/20240130043336/https://it.fawproject.com/) mediante software libero. Attualmente, continua ad essere sviluppato da altri sviluppatori.

[63] L'unico ruolo che viene svolto "in prima persona" da FIT è di tenere uniti i vari componenti di terze parti, che individualmente sono già largamente usati e collaudati.

[64] Ad esempio, Google offre Google Takeout per scaricare i dati relativi al proprio account (v. https://takeout.google.com/), e Apple permette di scaricare una copia dei dati collegati al proprio Apple ID (v. https://web.archive.org/web/20230903043707/https://support.apple.com/en-us/102208).

[65] In linea generale, ci si può fidare che i dati non vengano modificati dal gestore del servizio, perché non avrebbero nessun interesse a farlo. Il problema più rilevante è la quantità dei dati che è possibile ottenere in questo modo.

[66] Usa la licenza GPLv3. V. https://github.com/telegramdesktop/tdesktop.

[67] V. https://web.archive.org/web/20180827090156/https://telegram.org/blog/export-and-more.

[68] Analisi forense di memorie volatili, ossia, la RAM.

[69] Le istruzioni per il sequestro di dati informatici prevedevano lo spegnimento del computer, senza previa acquisizione della memoria. V. Amy L. Ayers, «Windows hibernation and memory forensics», Utica College ProQuest Dissertations Publishing, 2015, https://www.proquest.com/dissertations-theses/windows-hibernation-memory-forensics/docview/1676462584/se-2, p. 7.

[70] *Ibidem*, p. 1.

[71] A. Gammarota, *op. cit.*, p. 149.

L'acquisizione presenta numerose problematiche tecniche,[72] tra cui il fatto che il formato dei dati contenuto nelle memorie volatili non è documentato in maniera ufficiale, ed in ogni caso, cambia con frequenza.[73]

È sempre utile almeno provare ad acquisire i dati: nel caso peggiore saranno inutilizzabili o irrilevanti, ma nel caso migliore si potrebbero trovare informazioni o tracce utili per l'investigazione.[74] Per acquisire la RAM si può usare WinPmem[75] su Windows, e LinPmem[76] su Linux.

4.4 Software libero per conservare i dati

Dopo che i dati sono stati acquisiti, è necessario garantire la loro corretta conservazione.

Programmi come *BorgBackup*[77] e *Restic*[78] permettono di creare copie di backup dei dati, di proteggere i backup con la crittografia,[79] e di verificare la loro integrità.[80]

Le copie di backup possono essere conservate su *filesystem* specializzati per l'archiviazione dei file, come *OpenZFS*, che controlla automaticamente l'integrità dei

[72] Andrew Case, Golden G. Richard, «Memory forensics: The path forward», *Digital Investigation*, vol. 20, 2017, https://www.sciencedirect.com/science/article/pii/S1742287616301529, pp. 23–33.

[73] La documentazione è assente nel caso dei sistemi operativi proprietari, come Windows e macOS. Nei sistemi operativi che usano il kernel Linux, si può consultare il codice sorgente relativo alla gestione della memoria. Dato che la memoria è volatile, non è necessario "standardizzare" il suo formato, e assicurarsi che possa essere letto anche a distanza di tempo. V. Joe T. Sylve, Vico Marziale, Golden G. Richard, «Modern windows hibernation file analysis», *Digital Investigation*, vol. 20, 2017, https://www.sciencedirect.com/science/article/pii/S1742287616301487, pp. 16–22.

[74] Come ad esempio, la chiave crittografica per decrittare informazioni protette. V. Christopher Hargreaves, Howard Chivers, «Recovery of Encryption Keys from Memory Using a Linear Scan», *2008 Third International Conference on Availability, Reliability and Security*, 2008, pp. 1369–1376.

[75] V. https://github.com/Velocidex/WinPmem.

[76] V. https://github.com/Velocidex/Linpmem.

[77] V. https://web.archive.org/web/20240103151618/https://www.borgbackup.org/.

[78] V. https://web.archive.org/web/20240102073003/https://restic.net/.

[79] In modo da garantire la confidenzialità dei dati, anche nel caso di un *data breach*.

[80] In modo che sia possibile verificare che i dati non siano variati per *bit rot* o modifiche intenzionali da parte di terzi, anche a distanza di tempo.

dati.[81] La creazione di copie dei dati può essere effettuata con *Rsync*,[82] un programma per la copia di dati che usa un algoritmo *ad-hoc* per assicurare che la copia sia identica all'originale.[83]

La catena di custodia può essere redatta con *Git*. Il documento digitale che contiene la catena può essere redatto in qualsiasi formato.[84] Ogni volta che si aggiorna la catena di custodia, si crea un nuovo *commit*, il *commit* viene firmato digitalmente da tutti i partecipanti,[85] ed i partecipanti ottengono una copia[86] della catena digitale.[87]

4.5 Software libero per analizzare i dati

Per analizzare i dati, si possono seguire due strade.

La prima è di usare programmi di analisi integrati, come Autopsy.[88] I vantaggi sono che:

- Offrono un'interfaccia grafica unificata, che li rende più facili da usare, e permette di svolgere le operazioni tipiche in maniera efficiente;

[81] Per una guida alle funzioni di base di ZFS, v. Autori di ArchLinux Wiki, *ZFS/Virtual disks*, 2023, https://web.archive.org/web/20240202013900/https://wiki.archlinux.org/title/ZFS/Virtual_disks. La sez. 5 dimostra come ZFS può riparare i dati danneggiati.

[82] V. https://github.com/WayneD/rsync.

[83] In particolare, questo algoritmo ha due proprietà utili: il trasferimento dei file può essere interrotto e ripreso in un secondo momento senza dover ricominciare dall'inizio, e calcola l'hash dei dati durante il trasferimento, per garantire che l'originale e la copia contengano gli stessi dati. V. sez. "Rolling checksum" in Andrew Tridgell, Paul Mackerras, «The rsync algorithm», 1998, https://web.archive.org/web/20240124111006/https://rsync.samba.org/tech_report/tech_report.html.

[84] Git è in grado di gestire anche file binari (come un documento in formato .DOC, .DOCX, o .ODT), non soltanto file di testo.

[85] Git supporta nativamente le firme GPG (GNU Privacy Guard) per i *commit*, ma è possibile usare qualsiasi metodo diverso di firma digitale, purché la firma sia contenuta in un file che possa essere conservato all'interno del repository.

[86] La copia viene creata con *git clone*. Se i partecipanti hanno già una copia della catena, possono sincronizzare la loro copia con *git pull*.

[87] Distribuire copie dell'intera catena a più persone serve a garantire che esistano più copie di backup della catena in circolazione. Inoltre, se qualcuno prova ad alterare la catena, o se la catena viene danneggiata o persa per altri motivi, è possibile confrontare le copie in circolazione. Qualsiasi modifica risulterà in *repository* divergenti, pertanto è facile trovare le copie autentiche e non manipolate confrontando tutte le copie fra di loro: se almeno due o più copie sono uguali, salvo caso di collusione, quelle copie rappresentano la catena originale.

[88] V. https://github.com/sleuthkit/autopsy.

- Permettono di generare in maniera automatica un report finale delle operazioni svolte;
- In termini di capacità, sono paragonabili ai prodotti non-liberi.[89]

Lo svantaggio è che si è limitati dalle funzionalità del programma, e modificarlo potrebbe essere difficile.[90]

La seconda è di usare più programmi di analisi, separati e specializzati, per svolgere le analisi. Ad esempio, parte delle funzioni svolte da Autopsy può essere replicata usando altri programmi:

- *mount* può essere usato per aprire l'immagine forense dei supporti;[91]
- *md5sum* e *sha1sum* per calcolare l'hash di file;[92]
- *PhotoRec*[93] per cercare e recuperare file cancellati;
- *file*[94] per determinare il formato del file in analisi;
- *grep*,[95] *ripgrep*,[96] *ripgrep-all*[97] per eseguire ricerche di stringhe o espressioni regolari[98] all'interno di file e vari formati;

[89]Per un confronto tra Autopsy e altri prodotti commerciali, v. Dan Manson et al., «Is the Open Way a Better Way? Digital Forensics Using Open Source Tools», *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007. In ogni caso, anche se mancassero delle funzionalità rispetto al software proprietario, ci si può fidare in misura maggiore delle funzionalità che sono presenti, dato che è possibile studiare il loro funzionamento.

[90]Più il software è integrato, e più diventa necessario conoscere il suo esatto funzionamento per essere in grado di modificarlo adeguatamente. Le uniche alternative sono studiare il codice (che comporta un dispendio di tempo), oppure chiedere agli sviluppatori originali di introdurre le modifiche (che potrebbe comportare un dispendio di denaro, per incentivare gli sviluppatori ad accettare o dare priorità alla richiesta).

[91]Più precisamente, *mount* permette di montare una singola partizione. Se il disco contiene più partizioni, è necessario prima identificarle con *mmls*, che fa parte di *The Sleuth Kit* (v. https://github.com/sleuthkit/sleuthkit). V. https://web.archive.org/web/20240405181601/https://superuser.com/questions/562154/mounting-a-complete-disk-image-rescued-by-ddrescue/562158.

[92]Per calcolare l'hash di più file, è possibile usare i comandi *find* e *xargs*, v. https://web.archive.org/web/20240222053711/https://stackoverflow.com/questions/545387/linux-compute-a-single-hash-for-a-given-folder-contents/545413.

[93]V. https://web.archive.org/web/20240225063137/https://www.cgsecurity.org/wiki/PhotoRec

[94]V. https://github.com/file/file.

[95]V. https://web.archive.org/web/20240110035523/https://www.gnu.org/software/grep/.

[96]V. https://github.com/BurntSushi/ripgrep.

[97]V. https://github.com/phiresky/ripgrep-all.

[98]Le espressioni regolari permettono di cercare testo che corrisponde ad un certo modello. Per un'introduzione alle espressioni regolari, v. G. Moschitta, *Espressioni regolari: pattern, uso ed esempi*, 2006, https://web.archive.org/web/20200919192017/https://www.html.it/articoli/espressioni-regolari/.

- *xxd*[99] per visualizzare i file in formato esadecimale;
- *imagemagick*[100] ed *ffmpeg*[101] possono essere usati per estrarre miniature e anteprime di immagini e video;
- *stat* mostra la data di creazione, modifica e ultimo accesso per i file;
- *Wireshark* permette anche di analizzare i pacchetti che ha acquisito;
- *Volatility*[102] offre strumenti per analizzare la copia della memoria che è stata acquisita.

Il vantaggio di questi programmi è che possono essere combinati fra di loro all'interno di uno *script*,[103] che può essere:

- Generico, come nel caso di *OCFA*,[104] *CAINE*,[105] *sfdumper*;[106]
- Specifico per le operazioni da svolgere nel caso concreto.[107]

La maggiore flessibilità viene controbilanciata dalla maggiore difficoltà d'uso,[108] e il fatto che si sta creando una soluzione *ad-hoc*, invece di usarne una generica.[109]

Le macchine virtuali offrono la possibilità di eseguire un sistema operativo, chiamato *guest* (ospite) all'interno di un altro sistema operativo, detto *host*,[110] e possono trovare vari usi durante la fase dell'analisi.

[99] *xxd* viene usato dall'editor di testo *vim*. V. https://github.com/vim/vim/tree/00221487731ea1868c57259c7aa0eb713cd7ade7/src/xxd.

[100] V. https://web.archive.org/web/20240102055502/https://imagemagick.org/

[101] V. https://web.archive.org/web/20240102043106/https://ffmpeg.org/

[102] V. https://web.archive.org/web/20240311040001/https://volatilityfoundation.org/the-volatility-framework/.

[103] Un file che contiene una sequenza di comandi da eseguire.

[104] I moduli di OCFA avviano altre applicazioni per elaborare i dati, v. E. Huebner, S. Zanero, *op. cit.*, p. 59.

[105] È un'interfaccia grafica intorno ad una serie di programmi, v. E. Huebner, S. Zanero, *ivi*, pp. 75–78.

[106] V. E. Huebner, S. Zanero, *ivi*, pp. 117 ss., e https://web.archive.org/web/20080303035227/https://sfdumper.sourceforge.net/.

[107] In questo caso, lo *script* serve come documentazione per l'elenco delle operazioni che sono state compiute, ma permette di ripetere l'intera analisi semplicemente eseguendolo di nuovo.

[108] È necessario saper usare ogni singolo programma, che spesso non offrono interfacce grafiche, e sapere come scrivere *script* per il trattamento dei dati.

[109] Pertanto, è necessario dimostrare l'affidabilità dei singoli programmi usati, e la ragionevolezza dell'approccio seguito, mentre se si usa un programma integrato, basta solo dimostrare l'affidabilità del singolo programma.

[110] Per maggiori dettagli sulla nozione di virtualizzazione, ed un elenco del software libero disponibile, v. E. Huebner, S. Zanero, *op. cit.*, pp. 26–28.

In alcuni casi, il loro uso potrebbe essere necessario per esaminare a pieno l'immagine forense.[111]

In altri casi, il loro uso potrebbe risultare utile ai fini dell'analisi. È possibile usare contemporaneamente una distribuzione Linux (come *host*) e Windows (come *guest*).[112]

Ancora, è possibile duplicare le macchine virtuali, e creare delle *snapshot* del loro stato.[113]

Infine, le macchine virtuali possono essere utilizzate per svolgere esperimenti giudiziali, e ricostruire, in maniera controllata e ripetibile, lo svolgimento di fatti all'interno di un sistema informatico.[114]

Eseguendo più macchine virtuali in parallelo, diventa possibile anche ricostruire le dinamiche di un incidente informatico che ha coinvolto più sistemi.[115]

[111]Ad esempio, si pensi al caso in cui all'interno del sistema esistono dati protetti da crittografia, e si conosce la password. La soluzione più semplice ed affidabile per accedere a quei dati è di avviare il sistema, e decrittarli come lo farebbe un normale utente. Il sistema che è stato acquisito viene avviato come un sistema operativo *guest* all'interno del sistema operativo *host* con cui si esegue l'analisi, i dati vengono decrittati, e infine sono copiati dal sistema *guest* al sistema *host*, in modo che possano essere analizzati.

[112]Questo permette di avere accesso agli strumenti di analisi disponibili su entrambi i sistemi operativi, v. E. Huebner, S. Zanero, *op. cit.*, p. 40.

[113]V. E. Huebner, S. Zanero, *ivi*, p. 36. Se le operazioni di analisi vengono compiute all'interno di una macchina virtuale, è possibile duplicare l'ambiente di analisi, e quindi garantire la più completa riproducibilità e ripetibilità dell'analisi. Chiunque ottenga una copia della macchina virtuale otterrà anche una copia del sistema operativo e di tutti i programmi installati, ed è possibile avviarla in qualsiasi momento. Creare una *snapshot* (instantanea) permette di annullare tutte le modifiche successive alla sua creazione, e ritornare allo stato del sistema nel momento in cui era stata creata. Pertanto, si è liberi di provare analisi o eseguire modifiche potenzialmente distruttive, perché è sempre possibile tornare ad uno stato precedente.

[114]Come ad esempio, l'azione di un virus informatico sui dati dell'utente. V. A. Gammarota, *op. cit.*, p. 197.

[115]Ad esempio, la diffusione di un virus informatico all'interno di più macchine. V. A. Gammarota, *ivi*, pp. 197–198.

Conclusioni

La definizione ideale dell'informatica forense deve evidenziare che il trattamento dei dati informatici è finalizzato al loro uso all'interno di un processo, e deve essere formulata in modo che l'informatica forense trovi la massima applicazione possibile.

L'informatica forense si è sviluppata all'interno del diritto processuale penale. Con l'introduzione di reati dove il bene giuridico protetto sono i dati informatici in sé, diventava necessario sviluppare una disciplina che fosse in grado di ricostruire i fatti relativi ai dati informatici, e riformare il codice di procedura, che invece era stato pensato per fatti relativi al mondo materiale.

In Italia esiste una preferenza per le prove precostituite, che è particolarmente pericolosa per le prove informatiche, perché ostacola la loro acquisizione e valutazione all'interno del contraddittorio.

L'informatica forense incontra delle difficoltà importanti, perché il suo oggetto di studio è oscuro (non è sempre facile studiare i programmi, protocolli, formati, ecc.), instabile (la materia è in continua evoluzione) e fragile (i dati digitali possono essere modificati con facilità, senza lasciare tracce).

Tuttavia, piuttosto che considerare lo studio dei dati informatici una causa persa in partenza, è possibile usare quelle caratteristiche come un buon motivo per seguire un approccio scientifico e rigoroso nello studio dell'informatica forense.

Se l'uso dei dati informatici a fini probatori richiede l'uso di conoscenze scientifiche, allora è preferibile usare la perizia sia per acquisire, sia per valutare i dati informatici. I vantaggi sono che il trattamento dei dati informatici viene eseguito da persone con

conoscenze specifiche fin dall'inizio, e quindi c'è un minore rischio di danneggiare i dati, e se partecipano anche i consulenti tecnici, tutte le operazioni si possono svolgere in contraddittorio.

Il secondo capitolo dimostra come l'informatica forense è una disciplina che si presta ad essere regolata da più fonti. La struttura della prima parte del capitolo segue le fasi del trattamento della prova secondo gli standard ISO, ma il contenuto dei singoli capitoli richiama le norme contenute nel codice di procedura penale. Si è fatto riferimento al codice di procedura penale colombiano per analizzare una possibile disciplina della catena di custodia. Quanto richiesto dalla legge viene reinterpretato in base a quanto è possible fare con l'informatica.

La conclusione della prima parte è che esiste uno stretto legame fra la scienza, il processo, ed il software. Dato che la scienza ed il processo sono ispirati da principi simili (*peer-review* e contraddittorio, motivazione degli atti e verificabilità degli esperimenti, limitazione del segreti e pubblicazione dei risultati) è opportuno che principi analoghi si applichino anche al software che viene usato dagli esperti all'interno del processo.

Questo obiettivo è pienamente raggiunto con l'uso del software libero. In particolare, l'accesso al codice sorgente è necessario per soddisfare tutti e tre i principi, e la licenza GPL serve proprio a garantire non solo che il software già libero rimanga tale, ma anche che l'adozione della GPL si diffonda, a causa della interpretazione data da Stallman per quanto riguarda l'uso di componenti GPL all'interno di altri programmi.

Ancora, si dimostra che in generale, è preferibile usare il software libero, al posto del software proprietario, e che in alcuni casi l'apparente superiorità del software proprietario in realtà porta a costi e svantaggi nascosti. L'unico caso in cui è impossible usare il software libero è per lo sviluppo di captatori informatici, per il solo fatto che la loro modalità di funzionamento deve rimanere segreta. Ma il fatto che sia impossible conoscere il funzionamento del captatore aggiunge ulteriori dubbi sulla sua utilità come mezzo di prova.

Il terzo capitolo si concentra sugli aspetti più tecnici e pratici dello sviluppo del software libero, andando ad individuare elementi concreti che possono essere usati per argomentare a favore del suo uso e della sua affidabilità.

L'accesso al codice sorgente, che viene garantito dalle licenze libere, ha numerosi vantaggi, perché permette di esaminare:

- Come i programmatori hanno cercato di risolvere i problemi creati dalle caratteristiche sfavorevoli del linguaggio di programmazione usato;
- La documentazione relativa al codice stesso, e quindi di sapere come il programma funziona;
- Se e come il codice di terze parti è stato integrato all'interno del software;
- L'uso di strumenti automatizzati che permettono di analizzare e correggere problemi nel codice;
- L'uso di *tests* per garantire che il programma continua a funzionare correttamente, anche a seguito di modifiche del codice.

Nel caso di programmi proprietari, tutti questi elementi generalmente non vengono condivisi con il pubblico, perché sono strettamente legati allo sviluppo del software, e pertanto non possono essere nemmeno valutati dai tecnici.

L'uso di tecniche come la *containerization* e le *reproducible builds* permettono di garantire che il software funzioni sempre allo stesso modo, e sono utili per garantire la ripetibilità e riproducibilità delle analisi.

Il capitolo si conclude con le buone pratiche relative alle modalità di sviluppo del software.

Gli elementi fondamentali sono la definizione della funzione del programma, le linee-guida per la sua progettazione, e la scelta di una licenza libera.

L'uso di un sistema di controllo di versione è utile sia per gli sviluppatori (perché permette di sviluppare il software ed integrare le contribuzioni di terzi in maniera ordinata), sia per gli utilizzatori finali (perché permette di ottenere facilmente copie del

software, di verificare l'integrità della propria copia, di estrarre una versione specifica del programma in qualsiasi momento, e di valutare le differenze fra versioni).

Lo sviluppo del software libero deve essere ispirato alla massima trasparenza, e alla pubblicazione non solo del codice del programma, ma anche di qualsiasi altro elemento ad esso collegato.

Per quanto riguarda le contribuzioni di terze parti al codice libero, è importante definire un processo per la loro accettazione, che a sua volta deve essere trasparente. La recente *backdoor* trovata all'interno di *xz* non dimostra che il modello di sviluppo aperto a terzi è intrinsecamente inaffidabile, ma piuttosto, l'importanza di prendere precauzioni per evitare questo tipo di situazioni, e come è molto più semplice individuare, studiare e correggere i difetti nel software quando chiunque può ottenere una copia del codice sorgente del programma con facilità.

Il quarto capitolo indica alcuni esempi di software libero che sono già in uso.

Si enfatizza l'importanza di usare anche un sistema operativo libero, in modo da ridurre al minimo il numero di "scatole nere" di cui non si può conoscere o studiare il funzionamento in maniera agevole all'interno del processo di analisi. Esistono sistemi operativi liberi specializzati per l'informatica forense.

I sistemi operativi liberi generalmente permettono di installare il software da un archivio centralizzato, e gestito dagli stessi sviluppatori del sistema.

I problemi più importanti con l'installazione del software sono la tensione fra avere la versione più aggiornata del software e avere un sistema facilmente riproducibile, e la fiducia che si deve riporre in chi gestisce l'archivio. Per questi motivi, è preferibile usare un sistema operativo libero specializzato, che contiene software preinstallato.

Per quanto riguarda l'acquisizione dei dati, esiste un numero ristretto di programmi generalmente maturi, e usati da tempo. FIT rappresenta un'eccezione parziale, perché è un programma recente, ma che fa largo uso di software di terze parti largamente maturo.

Per quanto riguarda la conservazione dei dati, esistono programmi che permettono

di garantire la corretta conservazione e duplicazione dei dati. Le caratteristiche dello strumento di controllo della versione *Git* permettono di utilizzarlo anche per la redazione della catena di custodia.

Infine, per quanto riguarda l'analisi dei dati, si può scegliere fra due approcci. Il primo è di usare uno strumento di analisi integrata come Autopsy, che offre un'interfaccia grafica, ed è paragonabile in termini di funzionalità anche al software proprietario.

Il secondo è di usare strumenti di analisi separati e specializzati, e di combinare le loro funzionalità mediante un file che contiene le istruzioni da eseguire. Questo approccio è più complesso, ma è anche più flessibile e maggiormente riproducibile.

Le macchine virtuali non sono uno strumento di analisi in sé, ma sono estremamente utili per l'informatica forense, perché permettono di studiare il funzionamento di un sistema, di creare ambienti di analisi riproducibili, e di svolgere esperimenti giudiziali.

In conclusione, si può affermare che il software libero è il modello che risponde meglio alle esigenze dell'informatica forense. I problemi che limitano la sua adozione non sono problemi tecnici. Anche laddove abbia meno funzionalità del software proprietario è sempre possibile migliorarlo, ed in ogni caso è sempre possibile studiare e spiegare l'esatto funzionamento delle funzionalità già presenti.

Piuttosto, il vero ostacolo alla sua adozione sono dei problemi sociali. I tecnici non usano il software libero perché non sono al corrente della sua esistenza (gli sviluppatori del software libero generalmente non hanno le capacità economiche per pubblicizzarlo), per sfiducia (i tecnici ritengono che il software libero non sia sufficientemente sofisticato, o non si fidano del modello di sviluppo aperto) o per pura inerzia (sono abituati ad un certo programma, e non hanno un incentivo sufficiente per imparare ad usarne un altro).

In primo luogo, è necessario sensibilizzare i giuristi per quanto riguarda le esigenze relative al trattamento dei dati informatici, e come il software libero si allinea alle

esigenze processuali, in modo che possano fare pressione sui tecnici affinché usino strumenti di analisi liberi.

In seguito, è necessario sensibilizzare i tecnici per quanto riguarda i principi relativi alla prova nel processo, tra cui l'importanza del diritto alla difesa, e di un contraddittorio tecnico approfondito, che presuppongono la possibilità di sapere esattamente come funziona il software con cui si ricostruiscono i fatti.

In generale, è importante continuare a rendere il software libero più accessibile e facile da usare, in modo da incentivare la sua adozione, e re-implementare il software proprietario già esistente, come è successo con FIT[116] e FAW.

Infine, sarebbe utile menzionare l'uso del software libero nel codice di procedura, almeno negli istituti che riguardano l'acquisizione dei dati informatici. Ad esempio, quando si parla del sequestro di dati informatici, e si richiede l'uso di una "procedura che assicuri la conformità dei dati acquisiti a quelli originali" (art. 254-*bis* c.p.p.) si potrebbe inserire un'espressione come "laddove possibile, con l'uso di software con codice sorgente aperto".[117]

Purtroppo, il recente DDL sul sequestro di dati digitali che propone l'introduzione dell'art. 254-*ter* c.p.p.[118] non affronta il problema. Si continuano ad usare formule generiche[119] che non menzionano in maniera specifica la necessità di usare il software libero per l'acquisizione.

La volontà del legislatore di applicare il principio di proporzionalità all'interno

[116]V. nota 59 a p. 94.

[117]Volendo, si potrebbe richiedere anche "la consegna di una copia del programma informatico o sistema operativo usato durante l'acquisizione", "l'uso di tecniche per verificare la riproducibilità della compilazione" (le *reproducible builds*, per dimostrare che il programma non è stato alterato prima dell'acquisizione), aggiungere una sanzione processuale di inutilizzabilità per le prove informatiche non acquisite con l'uso di programmi informatici con codice sorgente aperto, ecc.

[118]Disegno di legge d'iniziativa dei senatori Zanettin e Bongiorno, comunicato alla presidenza il 19 luglio 2023, "Modifiche al codice di procedura penale in materia di sequestro di dispositivi e sistemi informatici, smartphone e memorie digitali", v. https://web.archive.org/web/20230725110052/https://www.giurisprudenzapenale.com/wp-content/uploads/2023/07/ddl-806__427387.pdf.

[119]Come "mediante una procedura che assicuri la conformità della copia all'originale e la sua immodificabilità" (art. 1 co. 2), e "con procedure che assicurino la conformità della copia ai dati fonte e l'immodificabilità della medesima" (art. 1 co. 6).

del sequestro[120] e l'introduzione di una sanzione di inutilizzabilità espressa (art. 1 co. 7) vanno lodate, ma si deve criticare la formulazione dei casi in cui si applica la disciplina[121] e l'espresso sfavore per la perizia nella procedura.[122]

[120]Si prevede che le operazione di selezione dei dati siano svolte in contraddittorio con gli interessati (co. 2).

[121]Nell'art. 1 co. 1 si menzionano "dispositivi e sistemi informatici", "smartphone", e "memorie digitali". Sarebbe stato sufficiente fermarsi alla prima espressione, dato che gli "smartphone" e le "memorie digitali" (si presuppone che si faccia riferimento ai supporti materiali che contengono dati digitali) rientrano già in quelle categorie.

[122]Nell'art. 1 co. 3 si prevede che le operazioni di selezione debbano essere svolte ai sensi dell'art. 360 c.p.p. ("Accertamenti tecnici non ripetibili"). La categorizzazione è corretta (l'acquisizione di un dispositivo comporta sempre il rischio di modificarlo), ma non si comprende perché il legislatore precluda categoricamente all'indagato la possibilità di richiedere una perizia. Sacrificare il diritto alla difesa per garantire che la procedura di sequestro si concluda in tempi rapidi è irragionevole e mette anche a rischio l'integrità dei dati.

Bibliografia

Note:

- Se una fonte è disponibile su internet, ma non è immediatamente disponibile al pubblico, è possibile visualizzarla usando l'accesso istituzionale dell'Università di Bologna.
- *Rivista di Diritto Processuale* è consultabile online: https://www.edicolaprofessionale.com/RDP.

Apple Inc., «Apple Platform Security», 2022. https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf.

Ayers, Amy L., «Windows hibernation and memory forensics», Utica College ProQuest Dissertations Publishing, 2015. https://www.proquest.com/dissertations-theses/windows-hibernation-memory-forensics/docview/1676462584/se-2.

Battiato, Sebastiano, Giuseppe Messina, Rosetta Rizzo, «Image forensics. Contraffazione digitale e identificazione della camera di acquisizione: status e prospettive», 2014. https://www.researchgate.net/publication/242495487.

Beekmans, Gerard, *Linux From Scratch. Version 12.0*, Bruce Dubbs (a cura di), 2023. https://web.archive.org/web/20230901165736/https://www.linuxfromscratch.org/lfs/view/stable/.

Bilaniuk, Stefan, «Is mathematics a science?», 1996. http://euclid.trentu.ca/math/sb/misc/mathsci.html.

Blachowicz, James, «How Science Textbooks Treat Scientific Method: A Philosopher's Perspective», *The British Journal for the Philosophy of Science*, vol. 60, fasc. 2, 2009, pp. 303–344. https://doi.org/10.1093/bjps/axp011.

Blomqvist, Jørgen, *Primer on International Copyright and Related Rights*, Edward Elgar Publishing, 2014.

Butler, Simon, Jonas Gamalielsson, Björn Lundell, Christoffer Brax, Anders Mattsson, Tomas Gustavsson, Jonas Feist, Bengt Kvarnström, Erik Lönroth, «On business adoption and use of reproducible builds for open and closed source software», *Software Quality Journal*, vol. 31, fasc. 3, settembre 2023, pp. 687–719. https://doi.org/10.1007/s11219-022-09607-z.

Caneschi, Gaia, «Le nuove indagini tecnologiche e la tutela dei diritti fondamentali. L'esperienza del captatore informatico», *Diritto Penale Contemporaneo*, fasc. 2, 2019, pp. 417–429. https://dpc-rivista-trimestrale.criminaljusticenetwork.eu/pdf/DPC_Riv_Trim_2_2019_caneschi.pdf.

Canestrari, Stefano et al., *Diritto penale. Lineamenti di parte speciale*, Monduzzi Editoriale, 2016.

Case, Andrew, Golden G. Richard, «Memory forensics: The path forward», *Digital Investigation*, vol. 20, 2017, pp. 23–33. https://www.sciencedirect.com/science/article/pii/S1742287616301529.

Casey, Eoghan, *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*, USA, Academic Press, Inc., 2011.

Cassazione Penale, Quarta Sezione, «Sent. n. 43786/2010», 2010. https://web.archive.org/web/20211128212823/https://olympus.uniurb.it/index.php?option=com_content&view=article&id=3919:cassazione-penale-sez-4-13-dicembre-2010-n-43786&catid=17&Itemid=138.

Cassazione Penale, Quinta Sezione, «Sent. n. 1801/2022», 2022. https://web.archive.org/web/20231222154506/https://www.italgiure.giustizia.it/xway/application/nif/clean/hc.dll?verbo=attach&db=snpen&id=./20220117/snpen@s50@a2022@n0180

1@tS.clean.pdf.

———, «Sent. n. 36080/2015», 2015. https://web.archive.org/web/20171104040843/https://www.giurisprudenzapenale.com/wp-content/uploads/2015/09/cass-pen-2015-36080.pdf.

Chacon, Scott, Ben Straub, «Pro Git. Version 2.1.413», 2023. https://web.archive.org/web/20231223152842/https://github.com/progit/progit2/releases/download/2.1.413/progit.pdf.

Cinti, Mariagrazia, «Quantificazione ed individuazione delle alterazioni dei dati nell'ambito di indagini di Informatica Forense», *Alma Mater Studiorum – Università di Bologna*, 2011. http://amslaurea.unibo.it/2736/.

Clarich, Marcello, *Manuale di diritto amministrativo*, Società editrice il Mulino, 2022.

Conso, Giovanni, Marta Bargis, Vittorio Grevi, *Compendio di procedura penale*, CEDAM, 2020.

Court of Appeals of District of Columbia, «Frye v. United States, 293 F. 1013 (D.C. Cir. 1923)», 1923. https://web.archive.org/web/20230202073721/https://nij.ojp.gov/sites/g/files/xyckuh171/files/media/document/frye-v-US.pdf.

Diaz, Antonio Diaz, «GNU ddrescue Manual», 2023. https://web.archive.org/web/20240109210952/https://www.gnu.org/software/ddrescue/manual/ddrescue_manual.html.

Dolstra, Eelco, «The purely functional software deployment model», Utrecht University, 2006. https://dspace.library.uu.nl/handle/1874/7540.

Dolstra, Eelco, Andres Löh, «NixOS: a purely functional Linux distribution», 2008. https://github.com/edolstra/edolstra.github.io/blob/2eed3fdbff656d01fe5372e9bf322799de0eaba7/pubs/nixos-icfp2008-submitted.pdf.

Feenberg, Daniel, «Can Intelligence Agencies Read Overwritten Data?», 2013. https://back.nber.org/sys-admin/overwritten-data-guttman.html.

Ferrazzano, Michele, «Indagini forensi in tema di scambio di file pedopornografici mediante software di file sharing a mezzo peer-to-peer», Alma Mater Studiorum –

Università di Bologna, 2014. http://amsdottorato.unibo.it/6697/.

Free Software Foundation, «GNU Coreutils», 2023. https://web.archive.org/web/20240205001115/https://www.gnu.org/software/coreutils/manual/html_node/index.html.

———, «GNU General Public License, Version 3, 29 June 2007», 2007. https://www.gnu.org/licenses/gpl-3.0-standalone.html.

———, «How to Choose a License for Your Own Work», 2022. https://web.archive.org/web/20220127041134/https://www.gnu.org/licenses/license-recommendations.html.

———, «The GNU C Library Reference Manual, for version 2.38», 2023. https://web.archive.org/web/20231227043035/https://sourceware.org/glibc/manual/pdf/libc.pdf.

———, «Various Licenses and Comments about Them», 2023. https://web.archive.org/web/20231018041504/https://www.gnu.org/licenses/license-list.html.

———, «What is Free Software?», 2023. https://web.archive.org/web/20231230224545/https://www.gnu.org/philosophy/free-sw.en.html.

Gabriel, Richard P., «Lisp: Good News, Bad News, How to Win Big», 2000. https://web.archive.org/web/20070706112430/https://www.dreamsongs.com/Files/LispGoodNewsBadNews.pdf.

Gammarota, Antonio, «Informatica forense e processo penale: la prova digitale tra innovazione normativa e incertezze giurisprudenziali», Alma Mater Studiorum – Università di Bologna, 2016. http://amsdottorato.unibo.it/7723/.

Geraci, Antonino, «I contratti di licenza d'uso del software», Università degli Studi di Parma, 2015. https://www.repository.unipr.it/handle/1889/2715.

Gutmann, Peter, «Secure Deletion of Data from Magnetic and Solid-State Memory», 1996. https://www.usenix.org/legacy/publications/library/proceedings/sec96/full_papers/gutmann/index.html.

Hargreaves, Christopher, Howard Chivers, «Recovery of Encryption Keys from Memory Using a Linear Scan[123]», *2008 Third International Conference on*

[123]https://doi.org/10.1109/ARES.2008.109

Availability, Reliability and Security, 1369–1376, 2008.

Holzmann, Gerard J., «The Power of 10: Rules for Developing Safety-Critical Code», 2006. https://ieeexplore.ieee.org/document/1642624.

Huebner, Eva, Stefano Zanero, *Open Source Software for Digital Forensics*, Springer Science+Business Media, 2010.

Leurent, Gaëtan, Thomas Peyrin, «SHA-1 is a Shambles: First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust», 2020. https://www.usenix.org/system/files/sec20-leurent.pdf.

Maldonato, Lucia, «Algoritmi predittivi e discrezionalità del giudice: una nuova sfida per la giustizia penale», *Diritto Penale Contemporaneo*, fasc. 2, 2019, pp. 401–416. https://dpc-rivista-trimestrale.criminaljusticenetwork.eu/pdf/DPC_Riv_Trim_2_2019_maldonato.pdf.

Manson, Dan, Anna Carlin, Steve Ramos, Alain Gyger, Matthew Kaufman, Jeremy Treichelt, «Is the Open Way a Better Way? Digital Forensics Using Open Source Tools[124]», *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)*, 2007.

Marinucci, Giorgio, Emilio Dolcini, Gian Luigi Gatta, *Manuale di Diritto Penale. Parte Generale. Nona edizione*, Giuffrè Francis Lefebvre, 2020.

Miller, Barton P., Mengxiao Zhang, Elisa R. Heymann, «The Relevance of Classic Fuzz Testing: Have We Solved This One?», *IEEE Transactions on Software Engineering*, vol. 48, fasc. 6, 2022, pp. 2028–2039. https://ieeexplore.ieee.org/abstract/document/9309406.

National Institute of Standards and Technology, «Secure Hash Standard (SHS)», 2015. http://dx.doi.org/10.6028/NIST.FIPS.180-4.

Nugent, Hugh, «State Computer Crime Statutes», 1991. https://www.ojp.gov/ncjrs/virtual-library/abstracts/state-computer-crime-statutes.

Open Source Initiative, «OSI Approved Licenses», 2024. https://web.archive.org/web/

[124]https://doi.org/10.1109/HICSS.2007.301

20240106034506/https://opensource.org/licenses/.

———, «The Open Source Definition», 2007. https://web.archive.org/web/20231231152615/https://opensource.org/osd/.

Ousterhout, John K., «Scripting: Higher Level Programming for the 21st Century», 1998. https://ieeexplore.ieee.org/document/660187.

Pasini, Manuele, «Programmazione memory-safe senza garbage collection: il caso del linguaggio Rust», *Alma Mater Studiorum – Università di Bologna*, 2019.

Raymond, Eric Steven, *The Art of Unix Programming*, Addison-Wesley, 2003. http://www.catb.org/esr/writings/taoup/html/.

Renzetti, Silvia, «La prova scientifica nel processo penale: problemi e prospettive», *Rivista di Diritto Processuale*, vol. 75, fasc. 2, 2015, pp. 399–423.

Stallman, Richard, «For Clarity's Sake, Please Don't Say "Licensed under GNU GPL 2"!», 2022. https://web.archive.org/web/20220219074031/https://www.gnu.org/licenses/identify-licenses-clearly.html.

———, «License Compatibility and Relicensing», 2021. https://www.gnu.org/licenses/license-compatibility.html.en.

———, «Linux and the GNU System», 2021. https://web.archive.org/web/20211109122924/http://www.gnu.org/gnu/linux-and-gnu.en.html.

———, «Selling Exceptions to the GNU GPL», 2021. https://www.gnu.org/philosophy/selling-exceptions.html.

Stallman, Richard, Bruno Haible, «Why CLISP is under GPL», 2000. https://gitlab.com/gnu-clisp/clisp/-/blob/master/doc/Why-CLISP-is-under-GPL.

Supreme Court of the United States, «Daubert v. Merrell Dow Pharmaceuticals, Inc., 509 U.S. 579 (1993)», 1993. https://web.archive.org/web/20221012193634/https://tile.loc.gov/storage-services/service/ll/usrep/usrep509/usrep509579/usrep509579.pdf.

Sylve, Joe T., Vico Marziale, Golden G. Richard, «Modern windows hibernation file analysis», *Digital Investigation*, vol. 20, 2017, pp. 16–22. https://www.sciencedirect.com/science/article/pii/S1742287616301487.

TAR Campania, Napoli, Sez. III, «Sent. n. 7003/2022», 2022. https://web.archive.org/web/20231222125832/https://portali.giustizia-amministrativa.it/portale/pages/istituzionale/visualizza?nodeRef=&schema=tar_na&nrg=202105119&nomeFile=20220703_01.html&subDir=Provvedimenti.

Thompson, Ken, «Reflections on trusting trust», *Commun. ACM*, vol. 27, fasc. 8, agosto 1984, pp. 761–763. https://doi.org/10.1145/358198.358210.

Tridgell, Andrew, Paul Mackerras, «The rsync algorithm», 1998. https://web.archive.org/web/20240124111006/https://rsync.samba.org/tech_report/tech_report.html.

Wolbe, Miles, «Can data be recovered from a zero-filled hard drive?», 2018. https://tinyapps.org/docs/recovering_data_from_zero_filled_hard_drive.html.

Wu, Quishi, Kangjie Lu, «On the Feasibility of Stealthily Introducing Vulnerabilities in Open-Source Software via Hypocrite Commits», 2021. https://web.archive.org/web/20210928192905/http://www.coding-guidelines.com/code-data/OpenSourceInsecurity.pdf.