Ponovljiva izgradnja programske podrške

Banjan, Matija

Master's thesis / Diplomski rad

2022

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka / Sveučilište u Rijeci**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:195:664724

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Diplomski studij Informatike, smjer Poslovna informatika

Banjan Matija

Ponovljiva izgradnja programske podrške

Diplomski rad

Mentor: doc. dr. sc. Vedran Miletić

Rijeka, 8. prosinca 2022.

Sažetak

Određeni softver moguće je ponovno izgraditi, to jest reproducirati ukoliko je od početka dostupan izvorni kod softvera, okruženje za izgradnju te uputstva za tu istu izgradnju te je softver predviđen za reprodukciju izgradnje. Imajući sve nužne podatke, svi specificirani artefakti se mogu rekreirati bajt po bajt. Relevantni atributi okruženja za izgradnju, upute za izgradnju i izvorni kod, kao i očekivani reproducirani artefakti definirani su od strane autora ili distributora. Artefakti gradnje dijelovi su rezultata izgradnje koji su željeni primarni rezultat.

Izvorni kôd je najčešće se navodi kao specifična revizija u nekom repozitoriju ili arhivirani izvorni kod. Relevantni atributi okruženja za izgradnju obično uključuju zavisnosti (dependencies) i njihove verzije, zastavice konfiguracije konstrukcije i varijable okruženja koliko ih koristi sam sustav izgradnje (npr. lokalne i regionalne postavke operacijskog sustava). Skup atributa poželjno je imati sveden na manju brojku. Artefakti koji nastaju prilikom izgradnje bi uključivali izvršne datoteke (executables), distribucijske pakete ili slike datotečnog sustava. Obično ne uključuju zapisnike izgradnje ili slične pomoćne izlaze koji se mogu razlikovati kod reprodukcije izgradnje.

Ponovljivost artefakata provjerava se usporedbom bajt po bajt. To se obično izvodi pomoću kriptografski sigurnih hash funkcija. Autori ili distributeri označavaju stranke koje tvrde da se ponovno može izgraditi skup artefakata. To mogu biti originalni autori, održavači distribucije ili bilo koji drugi distributeri.

Ključne riječi

izvorni kod softvera, izgradnja softvera, artefakti izgradnje softvera, okruženje izgradnje softvera, ponovljiva izgradnja softvera

Sadržaj

1. Uvod

Iako svatko može pregledati izvorni kod besplatnog i otvorenog softvera na zlonamjerne nedostatke, većina se softvera distribuira unaprijed sastavljena bez metoda za provjeru da li odgovaraju ili ne. To potiče napade na programere koji objavljuju softver, ne samo putem tradicionalnog iskorištavanja, već i u oblicima političkog utjecaja, ucjena ili čak prijetnji nasiljem.

To posebno zabrinjava programere koji surađuju na softveru za privatnost ili sigurnost, pošto napadi na njih obično dovode do ugrožavanja posebno politički osjetljivih meta poput disidenata, novinara i zviždača, kao i svih koji žele sigurno komunicirati pod represivnim režimima.

Iako su pojedinačni programeri prirodna meta, postoji i mogućnost napada na infrastrukturu izgradnje, pošto bi uspješan napad omogućio pristup velikom broju računalnih sustava korisnika softvera. Modificiranjem generiranih binarnih datoteka umjesto izmjene izvornog koda, nedopuštene promjene su praktički nevidljive kako za originalne autore, tako i za korisnike.

Motivacija projekta "Reproducible builds" je omogućiti provjeru da tokom procesa izgradnje softvera nisu uvedene dodatne ranjivosti ili stražnja vrata. Obećavajući da se iz istog izvora uvijek generiraju identični rezultati, omogućava svim trećim stranama da postignu konsenzus o "ispravnom" rezultatu, ističući sva odstupanja kao sumnjiva i vrijedna dodatnog ispitivanja.

Ova sposobnost primjećivanja je li programer ili sustav izgradnje ugrožen tada sprječava pojavljivanje takvih prijetnji ili napada jer se svako kompromitiranje softvera prilikom izgradnje putem programera ili infrastrukture može brzo otkriti. Zbog toga nema naročite motivacije za prijetnju pojedinačnim programerima ili prisilu na iskorištavanje i/ili razotkrivanje njihovih kolega.

Prvo, sustav izgradnje mora biti potpuno deterministički: transformacija datog izvora uvijek mora stvoriti isti rezultat. Na primjer, trenutni datum i vrijeme ne smiju se bilježiti, a izlaz uvijek mora biti zapisan istim redoslijedom.

Drugo, skup alata koji se koriste za izvođenje gradnje i općenitije samog okruženja za izgradnju trebao bi se zabilježiti ili unaprijed definirati.

Treće, korisnicima bi trebalo omogućiti način da ponovno stvore dovoljno blisko okruženje za izgradnju, izvedu postupak izgradnje i potvrde da se izlaz podudara s izvornom verzijom.

2. Motivacija

Rad na ponovljivim izgradnjama u početku može izgledati kao puno truda s malim dobitkom. Iako se ovo odnosi na mnoge vrste poslova povezanih sa sigurnošću, postoje neki dobri argumenti i svjedočanstva o tome zašto su ponovljive građe važne.

2.1. Odupiranje napadima

U ožujku 2015. The Intercept je iz „Snowden leaks-a" objavio sažetak govora na internoj konferenciji CIA-e 2012. o Strawhorse: Napad na MacOS i iOS Software Development Kit [1]. Sažetak jasno objašnjava kako su neimenovani istraživači stvarali modificiranu verziju XCodea koja bi - bez znanja programera - stavila vodeni žig ili umetnula špijunski softver u kompajlirane aplikacije.

Nekoliko mjeseci kasnije, pronađen je zlonamjerni softver nazvan "XcodeGhost" koji cilja programere kako bi ih natjerao da nesvjesno distribuiraju zlonamjerni softver ugrađen u iOS aplikacije. Palo Alto Networks to opisuje kao [2]:

„XcodeGhost je prvi zlonamjerni softver kompajlera u OS X. Njegov zlonamjerni kod nalazi se u Mach-O objektnoj datoteci koja je prepakirana u neke verzije Xcode instalacijskih programa. Ovi zlonamjerni instalacijski programi zatim su preneseni na Baiduovu uslugu dijeljenja datoteka u oblaku za korištenje od strane kineskih iOS/OS X programera"

Svrha reproducibilnih konstrukcija softvera je upravo odupiranje takvim vrstama napada. Ponovno prevođenje ovih aplikacija s čistim kompajlerom učinilo bi problem lako vidljivim, posebno s obzirom na veličinu dodanog opterećenja.

Kao što su Mike Perry i Seth Schoen objasnili u prosincu 2014. tijekom razgovora na 31C3, problematične promjene mogu biti suptilnije, a jedan bit može biti jedina stvar potrebna za stvaranje sigurnosne rupe koja se može iskoristiti i udaljeno, odnosno bez fizičkog pristupa računalu [3]. Seth Schoen također je demonstrirao zlonamjerni softver na razini kernela koji bi kompromitirao izvorni kod dok ga čita kompajler, bez ostavljanja ikakvih tragova na disku. Iako prema našim saznanjima takvi napadi nisu primijećeni u divljini, ponovljive izgradnje jedini su način da ih se rano otkrije.

2.2. Osiguranje kvalitete

Potrebni su redoviti testovi kako bi se osiguralo da se softver može reproducirati u različitim okruženjima. Debian i druge distribucije besplatnog softvera zahtijevaju da njihovi korisnici moraju biti u mogućnosti izgraditi softver koji distribuiraju. Takvi redoviti testovi pomažu u izbjegavanju neuspjeha izgradnje iz izvornih grešaka i mogu otkriti rijetke probleme izgradnje kao što su problemi s vremenom, uvjeti izgradnje ili izgradnje na koje utječe lokalizacija.

Okruženja izgradnje mogu se razviti nakon što projekt više nije primao velike pomake. Tijekom rada na Debianu identificirano je nekoliko problematičnih, ali teško uočljivih grešaka testiranjem nadogradnji u različitim okruženjima [4]. Primjeri toga bili bi: biblioteka je imala različito binarno sučelje aplikacije za svaku izgradnju, iskrivljene nizove zbog nepodudarnosti kodiranja, nedostajućih prijevoda ili mijenjanja ovisnosti.

Programerima također pomaže to što moraju razmišljati o okruženju izgradnje i njihovom odnosu s vanjskim softverom ili pružateljima podataka. Oslanjanje na vanjske izvore bez rezervnih planova može dugoročno uzrokovati ozbiljne probleme.

Reproducibilne izgradnje također omogućuju ponovno stvaranje odgovarajućih simbola za ispravljanje pogrešaka za distribuiranu izgradnju što može pomoći u razumijevanju problema u softveru koji se koristi u proizvodnji.

2.3. Manje binarne razlike

Imati reproducibilne izgradnje znači da će samo promjene u izvornom kodu ili okruženju gradnje (kao što je verzija prevoditelja/kompajlera) dovesti do razlika u generiranim binarnim datotekama. Ovo minimizira promjene u artefaktima što smanjuje zahtjeve za pohranu i mrežni promet za delta ažuriranja.

Sa sličnim artefaktima, testiranje se može usredotočiti na dijelove koji su se promijenili, a istovremeno zadržati povjerenje u nepromijenjeni kod. To može ubrzati osiguranje kvalitete i brzinu razvoja.

Promjene u sustavu izrade mogu se lako testirati s reproduciranim verzijama: ako su izlazni artefakti identični, promjene neće utjecati na ponašanje u vremenu izvođenja.

2.4. Povećana brzina razvoja

Pakete ovisnosti ne treba ponovno graditi i zavisne zadatke ne treba ponovno pokretati ako ponovna izgradnja paketa ne daje drugačije rezultate. To može značajno smanjiti vrijeme izrade i dovesti do bržih brzina razvoja i nižih troškova.

Brzine izgradnje također se mogu poboljšati pokazivanjem da unakrsna kompilacija daje isti rezultat kao izvorna kompilacija, a zatim izvođenjem većine unakrsnih kompilacija na bržim strojevima.

2.5. Kako mogu vjerovati svojem kompajleru

Uobičajeno pitanje vezano za reproducibilne izgradnje je kako je moguće znati nije li okruženje izgradnje ugroženo ako svi koriste iste binarne datoteke? Ili kako se može vjerovati da kompajler koji je upravo izgrađen nije bio ugrožen stražnjim vratima u kompajleru koji je korišten za njegovu izradu?

Potonji je poznat u akademskoj literaturi od „Reflections on trusting trust” Kena Thompsona objavljenog 1984 [5]. To je dokument koji se spominje u opisu govora o "Strawhorse" spomenutom ranije.

Tehnika poznata kao Diverse Double-Compilation, koju je formalno definirao i istražio David A. Wheeler, može odgovoriti na ovo pitanje [6]. Kratki sažetak načina rada: uzmu se dva kompajlera, jedan pouzdan i jedan koji se testira. Kompajler koji se testira se gradi dvaput, jednom sa svakim kompajlerom. Koristeći kompajlere stvorene iz ove međugradnje, kompajler koji se testira, ponovno se gradi. Ako je izlaz isti, tada imamo dokaz da tijekom kompilacije nisu umetnuta stražnja vrata. Da bi ova shema funkcionirala, izlaz konačnih kompilacija mora biti isti. Upravo zbog takvih slučajeva su ponovljive izgradnje korisne.

3. Povijest

Idea o ponovljivoj izgradnji prvi je puta implementirana početkom 1990 za GNU alate. Pojam se prvi puta spomenuo u Debian svijetu 2000.-e godine, te zatim eksplicitnije 2007.-e godine na mailing listi debian-devel, no oba puta reakcije zajednice nisu bile entuzijastične.

Interes za ponovljive izgradnje ponovno se pojavio sa pojavom Bitcoina. Korisnicima Bitcoina trebao je način da vjeruju da ne preuzimaju oštećen softver. Početne verzije Gitiana, sigurne metoda distribucije softvera orijentirana na kontrolu izvora, što znači da se mogu preuzeti pouzdane binarne datoteke koje je provjerilo više izvora, napisane su 2011. godine kako bi se riješio taj problem [7]. Gitian pokreće izgradnje pomoću virtualnih strojeva i Git alata.

Otkriće postojanja globalnog nadzora nad komunikacijama i podacima 2013. povećala su interes za ponovljivim izgradnjama još više [8]. Mike Perry radio je na tome da Tor preglednik bude reproduciran u strahu od "zlonamjernog softvera koji napada razvoj softvera i sam izgrađuje procese za distribuciju svojih kopija na desetke ili čak stotine milijuna strojeva u jednom, službeno potpisanom, trenutnom ažuriranju" [9].

Uspjeh u stvaranju tako velikog komada softvera reproducibilnim pokazao je da je izvedivo i za druge projekte. To je potaknulo Lunar da organizira raspravu na DebConf13 u srpnju 2013 [10, p.]. Iako je rasprava bila zakazana u zadnjim trenutcima, bilo je tridesetak sudionika koji su bili jako zainteresirani, među njima članovi tehničkog odbora i nekoliko drugih ključnih timova.

Nakon još nekih istraživanja tijekom konferencije, stvorena je wiki stranica na Debianovom sjedištu [11]. Početni pristup bio je navesti Debianovu zajednicu programera i doprinositelja da prihvate tu ideju tako što bi napravio pet paketa različitih održavatelja koji se mogu reproducirati. Međutim, brzo se pokazalo da prije nego što se riješi problem u lancu alata, neće biti moguće reproducirati niti jedan paket.

3.1. Prva masovna reprodroduciranja

Lunar je osmislio prve zakrpe za DebianPts:dpkg tokom kolovoza 2013.-e, što je omogućilo reproduciranje DebianPts:hello-a. Prvo reproduciranje na velikoj skali nedugo nakon toga vodio je David Suárez, s varijacijama u vremenu i putu izgradnje. Od 5240 izvornih paketa, 24% ih je identifirano kao reproducibilno [12]. Također, prva verzija "pametne" skripte za usporedbu izgradnji je napisana kako bi pomogla u pregledu razlika.

```
PACKAGE_A=$1
PACKAGE_B=$2

diffc() {
        diff -u0 <(echo "$@" | sed -e "s,PACKAGE,$PACKAGE_A," | sh) \
                 <(echo "$@" | sed -e "s,PACKAGE,$PACKAGE_B," | sh) |
             grep -Ev '^(@@ |--- |\+\+\+ )'
}

diffc 'sha1sum PACKAGE' > /dev/null

diffc 'ar tv PACKAGE'

for file in debian-binary control.tar.gz data.tar.xz; do
        diffc "ar p PACKAGE $file | sha1sum | sed -e s/-$/$file/"
done

diffc 'ar p PACKAGE control.tar.gz | tar -ztvf -'
ar p $PACKAGE_A control.tar.gz | tar -zvtf - | grep '^-' | while read flags user size date time file; do
        diffc "ar p PACKAGE control.tar.gz | tar -zxOf - $file"
done

diffc 'ar p PACKAGE data.tar.xz | tar -Jtvf -'
ar p $PACKAGE_A data.tar.xz | tar -Jvtf - | grep '^-' | while read flags user size date time file; do
        diffc "ar p PACKAGE data.tar.xz | tar -JxOf - $file" |
            sed -e "s,Binary files [^ ]* and [^ ]* differ,Binary file $file differ,"
done
```

Slika 1: Skripta za usporedbu [63]

Drugo masovno reproduciranje napravljeno je prije prezentacije u distro devroom-u na FOSDEM'14. Upotrijebljen je nešto drukčiji pristup što se tiče puteva izgradnje te je također imao binutils izgrađen u deterministčkom načinu. Od 6887 izvornih paketa, 67% je pronađeno reproducibilno [13].

Prezentacije je povećala interes za projektom, te aktivirala mailing listu stvorenu nekoliko mjeseci prije. Tomasz Buchert je napisao zakrpu za alat Lintian [14], koja je u konačnici omogućavala provjeru gzip datoteka, a Stéphane Glondu je radio na sortiranju logova i eksperimentiranju sa alternativama za poteškoće kod putanja izgradnje [15].

3.2. Kontrolne datoteke .buildinfo

Paralelno je razmatrano nekoliko pristupa gdje i kako spremiti okruženje za izgradnju. Primarna idea je bila korištenje kontrolne datoteke .changes putem zamjenske varijable, no umjesto toga, Guillem Jover je predložio dodavanje novih polja prosljeđivanjem --changes-option="-DBuild-Env=... u dpkg-buildpackage [16].

Problem u vezi s korištenjem .changes datoteka je taj što se promjene kod snimanja okruženja za izgradnju ne čuvaju u arhivi. Da bi se koristili kao način snimanja okruženja, morali bi se distribuirati u arhivu, što u konačnici je ispalo kao pogrešno razumijevanje njihove svrhe. Kako im naziv govori, kontrolne datoteke .changes predstavljaju promjene u arhivi. One su inherentno dizajnirane da budu prolazne.

Umjesto navedenoga, odlučeno je ići sa novom .buildinfo kontrolnom datotekom, koja bi se dodala u arhivu zajedno sa binarnim paketima, te ih na taj način prenosima referencirajući ih u .changes datoteci.

3.3. Stvaranje alata strip-nondeterminism

Prije početka DebConf14, tokom ponovnih izgradnji je dana izričita vremena oznaka izdvojena iz datoteke .changes. Međutim, tokom rasprave postignut je konsenzus da se datum posljednjeg unosa u datoteci debian/chanelog može koristiti kao referentna vremenska oznaka ukoliko i kada je to potrebno [17].

Navedeno je pripomoglo drugoj ideji: generičkom alatu koji bi naknadno obrađivao različite formate datoteka za uklanjanje vremenskih oznaka ili drugog izvora nedeterminizma. Andrew Ayer je preuzeo zadatak stvaranja alata strip-nondeterminism i pripadne biblioteke, napisavši ih u Perlu. Taj alat je imao zadatak uklanjanja nedeterminističkih informacija poput vremenskih oznaka i naručivanja datotečnih sustava iz različitih formata datoteka i arhiva. Prva objavljena verzija obrađivala je datoteke sa ekstenzijama gzip, zip, jar, javadoc i .a [18].

3.4. Odustajanje od putanja izgradnje

U početku se je pretpostavljalo da bi se varijacije koje se događaju prilikom izgradnje paketa iz različitih putanja izgradnje trebale eliminirati. Ovo se pokazalo teškim. Glavni problem koji je identificiran je da je puna putanja do izvornih datoteka zapisana u simbolima za otklanjanje pogrešaka ELF datoteka.

U prvom pokušaju korištena je opcija -fdebug-prefix-map koja omogućuje mapiranje trenutnog direktorija u kanonski u onome što se bilježi. Ali opcije prevoditelja također se zapisuju u datoteku za ispravljanje pogrešaka. Dakle, morao se udvostručiti s -gno-record-gcc-switches da bi se koristio za ponovljivost. Prva ponovna izgradnja velikih razmjera pokazala je da je također bilo teško točno odrediti koji je bio stvarni put izgradnje [19].

U drugom pokušaju korišten je debugedit koji koristi Fedora i drugi, koji je korišten za promjenu izvornih staza na kanonsku lokaciju nakon izgradnje. Nažalost, gcc piše debug nizove u hashtable. Debugedit neće promijeniti redoslijed u tablici nakon „krpanja" nizova, tako da je rezultat i dalje neponovljiv. Dodavanje ove značajke u debugedit izgledalo je teško. Još uvijek je bilo moguće omogućiti da pristup funkcionira prosljeđivanjem -fno-merge-debug-strings, ali to je prostorno zahtjevno. Druga ponovna izgradnja velikih razmjera koristila je potonji pristup. I dalje je bilo teško ispravno pogoditi početni put izgradnje. Stéphane Glondu je bio prvi koji je predložio korištenje kanonske staze izgradnje za rješavanje problema [20].

Tijekom rasprave na DebConf14 ponovno je razmotrena ideja te je zaključeno da je doista prikladno odlučiti se za kanonski put izgradnje [21]. Kanonski put izgradnje ima dodatnu pogodnost što olakšava korištenje paketa za ispravljanje pogrešaka: jednostavno je potrebno raspakirati izvor na pravo mjesto, te nije potrebna dodatna konfiguracija.

Konačno, dogovoreno je da se polje Build-Path doda u .buildinfo jer je olakšalo reprodukciju početne izgradnje ako bi se kanonska lokacija izgradnje promijenila.

3.5. Poboljšani alat za usporedbu

Nakon početnog učitavanja alata strip-nedeterminism i integracije još nekih promjena o kojima se raspravljalo tijekom DebConf14 u DebianPts:dpkg i DebianPts:debhelper, Lunar je eksperimentirao sa 172 temeljna paketa; 30% izgradnji paketa je reproducirano bez daljnjih izmjena.

Budući da su trenutni alati za razumijevanje razlika među verzijama spori i teški za čitanje, Lunar je napisao debbindiff. Zamijenio je neučinkovite skripte ljuske strukturiranim Pythonom koji je stvarao izlaz u formatu HTML.

3.6. Kontinuirana integracija

Krajem rujna 2014. Holger Levsen je počeo raditi na proširenju jenkins.debian.net za izvođenje kontinuirane integracije za ponovljivost izgradnje. Paketi iz razvojne verzije Debiana (sid) počeli su

se graditi i obnavljati. Ovo je u početku uvelo varijacije za vrijeme i redoslijed datoteka, a kasnije je prošireno na korištenje različitih korisnika, grupa, naziva domaćina i lokaliteta.

Rezultati su bili vidljivi putem nove web stranice reproducible.debian.net. Proces analize kvarova u ponovljivosti izgradnje sada se može lakše dijeliti. Pojavili novi suradnici koji su počeli pridonositi, razvrstavajući uobičajene probleme i dajući zakrpe.

U srpnju 2015. Vagrant dodaje mogućnost korištenja ARM ploča i procesora za testiranje ponovljivosti arhitekture armhf. Dodane su u Jenkins u kolovozu 2015., a do prosinca su gotovo svi paketi na armhf-u testirani barem jednom.

3.7. dpkg-genbuildinfo

Na prijelazu u 2015. prototip generatora .buildinfo zamijenjen je novom implementacijom prikladnom za pravilno uključivanje u dpkg. Prije su samo paketi koji koriste dh mogli generirati .buildinfo i stoga su se mogli smatrati reproducibilnima. Nakon ažuriranja eksperimentalnog lanca alata, promjena je omogućila postizanje oznake od 80% ponovljivih izvornih paketa [12].

4. Izvođenje ponovljive izgradnje

Kao što smo dosad vidjeli, ideja reproducibilnih/ponovljivih izgradnji je omogućiti bilo kome da potvrdi da tijekom procesa izgradnje nisu uvedeni nedostaci reproduciranjem bajt po bajt identičnih binarnih paketa iz danog izvora [22].

Postizanje ponovljivih izgradnji zahtijeva suradnju višestrukih uloga uključenih u proizvodnju softvera. Na malim projektima, sve ove uloge može nositi jedna osoba, ali pomaže dijeliti odgovornosti.

4.1. Dobivanje determinističkog sustava izgradnje

Kako bi se softveru omogućila reproducibilna izgradnja, izvorni kod ne smije sadržavati nekontrolirane varijacije u izlaznu verziju.

Proces će funkcionirati bolje ako se takve varijacije otkriju prije nego što se korisnici suoče s binarnim datotekama koje nisu reproducibilne. Postavljanje testnog protokola u kojem se ponovna izgradnja izvodi prema varijacijama u okruženju (aspekti poput vremena, korisničkog imena, CPU-a, verzije sustava, datotečnih sustava, između mnogih drugih) će uvelike pomoći.

4.2. Definiranje okruženja za izgradnju

Budući da će različite verzije alata za kompilaciju vjerojatno proizvesti različite rezultate, korisnici moraju moći ponovno stvoriti okruženje za izgradnju dovoljno blisko izvornoj verziji. Nije potrebno da sam lanac alata bude identičan bajt po bajt, već i njegov izlaz mora ostati isti. Okruženje za izgradnju može se definirati dok se softver razvija ili se može definirati za vrijeme izgradnje.

4.3. Distribucija okruženja za izgradnju

Korisnici moraju biti u mogućnosti znati koje okruženje za izgradnju treba postaviti za ponovnu izgradnju softvera. Ako je okruženje za izgradnju definirano unaprijed i dio je izvornog koda, tada nisu potrebni nikakvi daljnji koraci.
U drugim slučajevima, okruženje izgradnje mora biti dostupno uz binarne datoteke. Idealan oblik je opis koji mogu razumjeti i ljudi i strojevi kako bi se omogućila automatska provjera, dok se ljudima

omogućuje provjera je li okolina zdrava.

4.4. Pružanje protokola usporedbe

Korisnici moraju imati jednostavan način za ponovno stvaranje okruženja za izgradnju, dobivanje izvornog koda, izvođenje izgradnje i usporedbu rezultata.
U idealnom slučaju, protokol usporedbe za provjeru identičnosti rezultirajućih binarnih datoteka trebao bi biti jednostavan. Usporedbu bajtova ili kriptografskih hash vrijednosti jednostavno je napraviti i razumjeti [23].
Druge tehnologije mogu zahtijevati uklanjanje kriptografskih potpisa ili ignoriranje određenih dijelova. Takve operacije moraju biti i dokumentirane i skriptirane. Recenzenti moraju lako razumjeti obrazloženje i kod.

4.5. Postizanje determinističkih izgradnji

Varijabla SOURCE_DATE_EPOCH je standardizirana varijabla okruženja koju distribucije mogu postaviti centralno i imati alate za izgradnju koji to koriste kako bi proizveli ponovljivi izlaz.
U praksi SOURCE_DATE_EPOCH specificira posljednju modifikaciju nečega, obično izvornog koda, (mjereno) u sekundama od Unix epohe, što je 1. siječnja 1970., 00:00:00 UTC [24].

Prije nego što ovo implementirate, trebali biste pregledati naš kontrolni popis kako biste vidjeli možete li izbjeći implementaciju.

Kontrolni popis [25]:

1. Hoće li se strip-nondeterminsm riješiti razlika umjesto vas ili je jednostavno dodati ovu funkcionalnost njemu. Ako da, onda vam ova varijabla nije potrebna.
2. Možete li zakrpati alat koji generira ove informacije, da jednostavno ne generira te informacije. Ako je ovo dobra ideja i održavatelj prihvaća vašu ideju, onda vam ova varijabla nije potrebna.
3. Možete li zakrpati alat koji generira ove informacije da biste umjesto toga preuzeli ovo iz unosa koji se lakše reproducira i koji je već prisutan u bilo kojem nadređenom sustavu izgradnje čiji se dio očekuje. Ako je to slučaj i održavatelj prihvaća vašu ideju, onda vam ova varijabla nije potrebna.
4. Možete li zakrpati alat koji generira ove informacije da umjesto toga preuzme ovo iz

standardnih varijabli okruženja koje smo dizajnirali, a koje bi vaša distribucija centralno postavila na ponovljive vrijednosti.

5. Možete li zakrpati paket koji koristi alat kako biste mu dali neke opcije specifične za alat kao što su CLI argumenti ili njegove vlastite varijable okruženja, kako biste izbjegli generiranje ovih informacija ili generirali ponovljivu vrijednost.

4.5.1. Postavljanje varijable SOURCE_DATE_EPOCH

U idućim ćemo naslovima ukratko pojasniti postavljanje varijable kroz nekoliko primjera.

4.5.1.1. Debian

U Debianu, ovo je automatski postavljeno na isto vrijeme kao i zadnji unos u debian/changelog, tj. isto kao izlaz dpkg-parsechangelog -SDate [26].

1. Za pakete koji koriste debhelper, verzije >= 9.20151004 (Bug:791823) izvoze ovu varijablu tijekom izgradnje, tako da potencijalno se ne mora ništa mijenjati. Jedna iznimka je ako određeni debian/rules treba ovu varijablu u dijelovima koji nisu debhelper, u kojem slučaju možete pokušati (3) ili (4).
2. Za pakete koji koriste CDBS, verzije >= 0.4.131 (Bug:794241) izvoze ovu varijablu tijekom izgradnje, tako da nisu potrebne nikakve promjene.
3. S dpkg >= 1.18.8 (Bug:824572) možete uključiti /usr/share/dpkg/pkg-info.mk ili /usr/share/dpkg/default.mk.
4. Ako nijedna od gornjih opcija nije dobra tada održavatelji paketa mogu postaviti i izvoziti ovu varijablu ručno u debian/rules:

   ```
   export SOURCE_DATE_EPOCH ?= $(shell dpkg-parsechangelog -STimestamp)
   ```

Ako trebate/želite podržati dpkg verzije starije od 1.18.8:

```
export SOURCE_DATE_EPOCH ?= $(shell dpkg-parsechangelog -SDate | date -f- +%s)
```

Ako trebate/želite podržati dpkg verzije starije od 1.17.0:

```
export SOURCE_DATE_EPOCH ?= $(shell dpkg-parsechangelog | grep -Po '^Date: \K.*'
| date -f- +%s)
```

4.5.1.2. Git

Za postavljanje SOURCE_DATE_EPOCH na posljednju izmjenu git repozitorija možete

koristiti 'git log', na primjer u ljusci:

```
SOURCE_DATE_EPOCH=$(git log -1 --pretty=%ct)
```

4.5.1.3. Python

```
import os
import time
date_str = time.strftime(
      "%Y-%m-%d",
      time.gmtime(int(os.environ.get('SOURCE_DATE_EPOCH', time.time())))
)
```

4.5.1.4. Bash / POSIX ljuska

Za GNU sustave:

```
BUILD_DATE="$(date --utc --date="@${SOURCE_DATE_EPOCH:-$(date +%s)}" +%Y-%m-%d)"
```

4.5.1.5. Perl

```
use POSIX qw(strftime); my $date = strftime("%Y-%m-%d",
gmtime($ENV{SOURCE_DATE_EPOCH} || time));
```

4.5.2. Deterministički sustavi izgradnje

Softver se ne može lako izgraditi reproducibilno ako izvor varira ovisno o čimbenicima koje je teško ili nemoguće kontrolirati poput redoslijeda datoteka u datotečnom sustavu ili trenutnog vremena [27].

Koji aspekt sustava izgradnje treba učiniti deterministčkim duboko je povezan s onim što je definirano kao dio okruženja izgradnje.
Primjerice, pretpostavljajući da će različite verzije prevoditelja proizvesti različite izlazne podatke, pa je upotreba određene verzije prevoditelja obavezna kao dio okruženja za izgradnju. Ista pretpostavka ne vrijedi nužno za jednostavnije alate kao što su grep ili sed gdje zahtjevi za okruženjem mogu biti labavi kao "bilo koji nedavni sustav sličan Unixu".
Ali teško da je dobra ideja ovlastiti da se sistemski generator pseudoslučajnih brojeva inicijalizira s danom vrijednošću prije izvođenja izgradnje, stoga je bolje da slučajnost ne utječe na izlaz izgradnje.
Još jedan konkretan primjer o tome gdje povući crtu: nema potrebe brinuti se o tome da sustav izgradnje daje stalan izlaz kada se izvodi u različitim putanjama izgradnje kada se put izgradnje smatra dijelom okruženja izgradnje, i stoga zahtijeva da se ponovne izgradnje izvode u isti direktorij kao izvorna verzija.

Osnove o tome kako napraviti sustav izgradnje determinističkim mogu se sažeti kao [27]:

1. Osigurajte stabilne ulaze.
2. Osigurajte stabilne rezultate.
3. Hvatajte što manje iz okoline.

4.5.3. Nestajanje nepostojanih ulaza

Unosi s mreže - čak i ako se ne čini tako - su nepostojani. Najbolje je napraviti sustav izgradnje koji se ne oslanja na udaljene podatke.
Ako mora biti slučaj, tada [28]:

1. osugirajte integritet korištenjem kriptografskih kontrolnih zbrojeva,
2. čuvajte sigurnosne kopije.

U idealnom slučaju, rezervna lokacija trebala bi biti dostupna uz sigurnosne kopije.

Dobar primjer je kako rade preneseni softver za FreeBSD (tzv. sustav portova). Opisi portova sadrže popis MASTER_SITES, popis datoteka koje treba dohvatiti u DISTFILES i distinfo datoteku s kriptografskim kontrolnim zbrojevima za svaku od ovih datoteka. FreeBSD infrastruktura osigurava da kopije svih dist datoteka budu dostupne na zrcalnoj mreži. Prilikom izgradnje porta, datoteke će se preuzeti odatle ako izvorno glavno mjesto nije dostupno.

4.5.4. Stabilan pored za unose

Ako izrada vašeg softvera zahtijeva obradu nekoliko ulaza odjednom, provjerite je li redoslijed stabilan među izradama.

Tipičan primjer je stvaranje arhive iz sadržaja imenika. Većina datotečnih sustava ne jamči da će popis datoteka u direktoriju uvijek rezultirati istim redoslijedom.

4.5.4.1. Makefile

Sljedeća slika Makefile rezultirat će nereproducibilnim verzijama:

```
SRCS = $(wildcard *.c)
tool: $(SRCS:.c=.o)
	$(CC) -o $@ $^
```

Slika 2: Makefile

Rješenja:

1. Eksplicitno navedite sve ulaze i osigurajte da će biti obrađeni tim redoslijedom.

```
SRCS = util.c helper.c main.c
tool: $(SRCS:.c=.o)
	$(CC) -o $@ $^
```

Slika 3: Rješenje a

2. Sortirajte ulaze

```
SRCS = $(sort $(wildcard *.c))
tool: $(SRCS:.c=.o)
	$(CC) -o $@ $^
```

Slika 4: Sortiranje ulaza

4.5.4.2. Problemi vezani uz lokalizaciju

Prilikom sortiranja ulaza, morate osigurati da na redoslijed sortiranja ne utječu lokalne postavke sustava. Neke lokalne oznake neće razlikovati velika i mala slova.

Na primjer, tar će prema zadanim postavkama koristiti redoslijed datotečnog sustava prilikom spuštanja direktorija:

```
$ tar -cf archive.tar src
```

Slika 5: tar

Rješenje je koristiti find i sort, ali sljedeće bi i dalje moglo imati razlike kada se izvodi pod različitim lokalitetima:

```
$ find src -print0 | sort -z |
    tar --no-recursion --null -T - -cf archive.tar
```

Slika 6: find i sort

Lokalizacija koja se koristi za sortiranje datoteka mora biti navedena kako bi se izbjegla

iznenađenja:

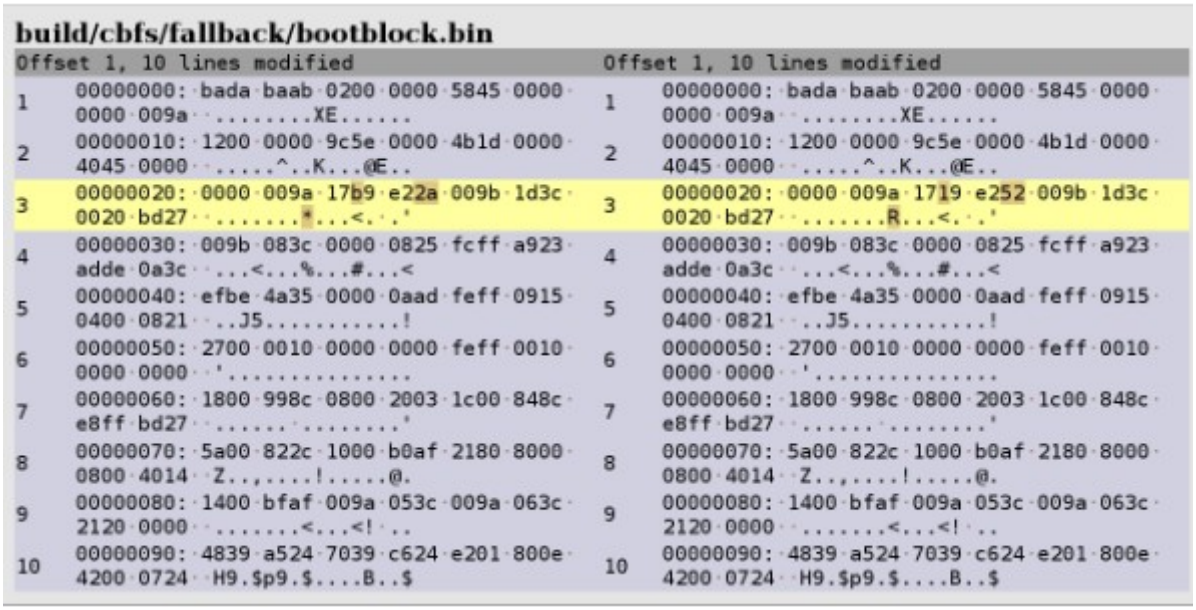
```
$ find src -print0 | LC_ALL=C sort -z |
    tar --no-recursion --null -T - -cf archive.tar
```

Slika 7: specificiranje lokalizacije

Ovo možda nije jedina promjena potrebna za Tar i druge arhivske formate jer oni obično imaju ugrađeno više problema s metapodacima.

4.5.5. Inicijalizacija vrijednosti

U jezicima koji ne inicijaliziraju vrijednosti, to se mora eksplicitno učiniti kako bi se izbjeglo hvatanje slučajnih bajtova u memoriji prilikom pokretanja.
Primjer preuzet iz coreboot-a:



Slika 8: Coreboot

Kod je pisao strukture podataka izravno bez inicijalizacije svih njezinih polja. Popravak je bio prilično jednostavan nakon što je identificiran:

```
--- a/util/bimgtool/bimgtool.c
+++ b/util/bimgtool/bimgtool.c
@@ -160,7 +160,7 @@ static const struct crc_t crc_type = {
 static int write_binary(FILE *out, FILE *in, struct bimg_header *hdr)
 {
        static uint8_t file_buf[MAX_RECORD_BYTES];
-       struct bimg_data_header data_hdr;
+       struct bimg_data_header data_hdr = { 0 };
        size_t n_written;

        data_hdr.dest_addr = hdr->entry_addr;
```

Slika 9: bingtool

Korištenje instrumentacijskih alata koji mogu otkriti takve slučajeve poput Valgrinda trebalo bi pomoći u identificiranju takvih problema. [29]

4.5.6. Informacije o verziji

Informacije o verziji ugrađene u softver moraju biti determinističke [30]. Protuprimjeri koriste trenutni datum ili inkrementalni brojač izgradnje.

Datum i vrijeme same izgradnje jedva da su od vrijednosti jer se stari izvorni kod uvijek može kompajlirati dugo nakon što je objavljen. Najbolje je kada informacije o verziji daju dobar pokazatelj koji je izvorni kod napisan.

Broj verzije može doći iz namjenske izvorne datoteke, dnevnika promjena ili iz sustava kontrole verzija. Ako je potreban datum, može se izdvojiti iz dnevnika promjena ili sustava kontrole verzija.

4.5.6.1. Kontrolni zbrojevi u Gitu kao identifikatori verzija softvera

Kriptografske kontrolne sume iz sustava kontrole revizije mogu se koristiti za identifikaciju izvornog sadržaja. Git commit ID-ovi su stoga dobar kandidat za uključivanje kao dio informacija o verziji.

Međutim, skraćeni Git hash identifikatori (kao što su oni dobiveni putem git describe ili git rev-parse) mogu biti izvor neponovljivosti. To je zato što broj heksadecimalnih znakova u skraćenom hash-u ovisi o broju objekata u Git repozitoriju.

Broj objekata ne samo da će se promijeniti tijekom vremena (zbog drugih obveza, čak i onih koji nisu u primarnoj razvojnoj grani), već će se također dramatično promijeniti ako se napravi 'plitki' klon (detalji se mogu pronaći u stranici priručnika git-clone(1) [31]) – ovi po dizajnu imaju manje

objekata.

Stoga se preporučuje da se odredi fiksno (ili "ne") skraćivanje prilikom dobivanja identifikatora korištenjem, na primjer, git describe --abbrev=12, git rev-parse --short=12 HEAD ili core.abbrev config postavljanje [32].

4.5.7. Vremenske oznake

Vremenske oznake najveći su izvor problema s reproducibilnošću [33]. Mnogi alati za izradu bilježe trenutni datum i vrijeme. Datotečni sustav ima, a većina arhivskih formata rado će zabilježiti vremena izmjene povrh vlastitih vremenskih oznaka. Također je uobičajeno zabilježiti datum izgradnje u samom softveru.

Često se vrijeme izgradnje koristilo kao približan način da se zna koja je verzija izvora izgrađena i koji su alati korišteni za to. Uz reproducibilne izgradnje, bilježenje vremena gradnje postaje besmisleno: s jedne strane, izvorni kod treba pratiti točnije nego samo vremensku oznaku, a s druge strane, okruženje gradnje treba definirati ili opsežno bilježiti.

Ako je potreban datum kako bi se korisnicima dala ideja o tome kada je softver napravljen, bolje je koristiti datum koji je relevantan za izvorni kod umjesto za međugradnju: stari softver se uvijek može kasnije izgraditi. Kao i informacije o verziji, najbolje je izvući takav datum iz sustava kontrole revizija ili iz dnevnika promjena.

4.5.7.1. Vanjski alati

Neki alati koji se koriste u procesima izrade, poput generatora koda ili dokumentacije, zapisuju vremenske oznake koje će stvoriti proizvode za izradu koji se ne mogu ponoviti.

Reproducible Builds predložio je varijablu okruženja SOURCE_DATE_EPOCH za rješavanje problema [24]. Alati koji ga podržavaju koristit će njegovu vrijednost—broj sekundi od 1. siječnja 1970., 00:00 UTC—umjesto trenutnog datuma i vremena (kada su postavljeni). Varijabla je formalno definirana u nadi da će se usvojiti šire.

Promjene potrebne za podršku SOURCE_DATE_EPOCH obično su prilično male i lako ih je napisati. Zakrpe za alate koji još ne podržavaju varijablu okruženja obično su dobro prihvaćene i pomažu svim korisnicima koji žele reproduktivne izgradnje.

U slučaju kada to nije moguće, opcija je napraviti naknadnu obradu na izlazu. Ideja je ili potpuno ukloniti vremenske oznake ili ih normalizirati na unaprijed određeni datum i vrijeme. Alat strip-

nondeterminism dizajniran je kao proširiv program za izvođenje takve normalizacije na različitim formatima datoteka.

Druga mogućnost je pokrenuti ove alate pomoću libfaketime [34]. Ova biblioteka se učitava kroz varijablu okoline LD_PRELOAD i presretat će pozive funkcija za dohvaćanje trenutnog doba dana. Umjesto toga će odgovoriti unaprijed definiranim datumom i vremenom. U nekim slučajevima radi savršeno i može riješiti probleme bez potrebe za velikim promjenama u danom sustavu izgradnje. Ali ako se bilo koji dio procesa izrade oslanja na vremenske razlike, stvari će poći po zlu. Jedan slučaj loše interakcije između libfaketimea i paralelne kompilacije identificiran je kao izvor problema s reproducibilnošću u pregledniku Tor [35].

4.5.8. Vremenske zone

Osim ako vremenska zona izgradnje nije dio okruženja izgradnje, mora se paziti da se dobije izlaz izgradnje kada se izgradnja izvodi u dvije različite vremenske zone.

Neki alati će napisati datum i vrijeme s povezanom vremenskom zonom:

```
$ LC_ALL=C date -d'@2147483647' --rfc-2822
Tue, 19 Jan 2038 04:14:07 +0100
```

Slika 10: Asocirana vremenska zona

Unatoč tome što mu je dano unaprijed određeno vrijeme u obliku Unix vremena (također se naziva epoha), ovaj izlaz ne bi se mogao reproducirati u vremenskoj zoni različitoj od UTC+0100. Jednostavno rješenje je postaviti potrebnu varijablu okruženja kako bi se alati prisilili da koriste UTC kao vremensku zonu:

```
$ LC_ALL=C date -u -d'@2147483647' --rfc-2822
Tue, 19 Jan 2038 03:14:07 +0000
```

Slika 11: UTC

Kada ne postoji namjenska opcija, najčešće je moguće postaviti varijablu okoline TZ:

```
$ TZ=UTC LC_ALL=C date -d'@2147483647' --rfc-2822
Tue, 19 Jan 2038 03:14:07 +0000
```

Slika 12: TZ

S tim u vezi zabrinutost je za formate koji ne sadrže informacije o vremenskoj zoni. Zip arhive su dobar primjer: uvijek se mora koristiti ista vremenska zona za njihovo raspakiranje kako bi se spriječile varijacije u vremenu izmjene datoteke.

4.5.8.1. Datum u obliku niza znakova s informacijama o vremenskoj zoni

Vremenski nizovi poput "Srijeda, 21. listopada 2015. 11:18:50" sami su po sebi dvosmisleni. Koju vremensku zonu koristi? Kako to treba razumjeti?

U kontekstu reproducibilnih izgradnji, najbolje je da svi vremenski nizovi sadrže informacije o vremenskoj zoni. Zamjenska opcija je pretpostaviti da su svi navedeni kao UTC [36].

Ako se vremenski nizovi bez specifikacije vremenske zone analiziraju u vremenskoj zoni sustava za izgradnju, može se dogoditi teško razumljivo ponašanje. Primjer je izračunavanje vremenske razlike u vremenskim zonama s različitim promjenama ljetnog računanja vremena. Kako različite vremenske zone imaju različita pravila, korisnik može dobiti više ili manje sati ovisno o vremenskoj zoni koja se koristi za izvođenje izgradnje.

4.5.9. Lokaliteti

Lokalizacija sustava izgradnje može utjecati na proizvode izgradnje. Iako je važno da programeri imaju pristup porukama o pogreškama na jeziku po vlastitom izboru, alati na čiji izlaz utječe trenutna lokalizacija mogu učiniti lokalizaciju izvorom problema s obnovljivošću.
Postoji mnogo aspekata koji se tiču lokaliteta. Oni koji slijede najvažniji su za razmatranje u kontekstu reproducibilnih izgradnji.

4.5.9.1. Vremenski format

Nekoliko uobičajenih funkcija oblikovanja vremena imat će izlaz ovisno o trenutnoj lokalnoj postavci. Na POSIX sustavu formatiranje će ovisiti o varijabli okruženja LC_CTIME, koju može nadjačati LC_ALL [37].
Za sustave izgradnje, najbolje je koristiti LC_ALL izravno:

```
$ LC_ALL=C date -u -d '2015-10-21'
Wed Oct 21 00:00:00 UTC 2015
```

Slika 13: Vremenski format

Vremenska zona sustava i varijabla okruženja TZ također će utjecati na izlaz funkcija formatiranja

vremena.

4.5.9.2. Redoslijed slaganja

Uobičajene funkcije sortiranja su pod utjecajem varijable okruženja LC_COLLATE, koju može nadjačati LC_ALL. Neki lokaliteti mogu biti prilično iznenađujući; to se obično prikazuje kada koristite sortiranje. Lokalitet fr_FR sortirat će se neovisno o veličini slova:

```
$ echo B a c | tr ' ' '\n' | LC_ALL=fr_FR.UTF-8 sort
a
B
c
```

Slika 14: fr_FR

C lokalizacija će sortirati prema vrijednostima bajtova i uvijek je dostupna:

```
$ echo B a c | tr ' ' '\n' | LC_ALL=C sort
B
a
c
```

Slika 15: c lokaliyacija

4.5.9.3. Zadano kodiranje znakova

Zadano sistemsko kodiranje znakova utjecat će na ulaz i izlaz mnogih alata. Definiran je pomoću varijable okoline LC_CTYPE, a može se nadjačati i pomoću LC_ALL.

Primjera kada koristite lynx za pretvaranje HTML dokumentacije u tekst:

```
LC_ALL=fr_FR lynx -dump -width 72 docs.html | file -
/dev/stdin: ISO-8859 text
```

Slika 16: lynx

C.UTF-8 pseudo-lokalitet uvijek se može koristiti za dobivanje zadanih nizova s UTF-8 izlazom:

```
LC_ALL=C.UTF-8 lynx -dump -width 72 docs.html | file -
/dev/stdin: UTF-8 Unicode text
```

Slika 17: C.UTF-8

4.5.10. Arhiviranje metapodataka

Većina arhivskih formata bilježi metapodatke koji će zabilježiti pojedinosti o okruženju izgradnje ako se ne obrati pažnja. Vrijeme zadnje izmjene datoteke je očito, ali redoslijed datoteka, korisnici, grupe, numerički ID-ovi i dopuštenja također mogu biti zabrinjavajući. Tar će se koristiti kao glavni primjer, ali ovi se savjeti odnose i na druge formate arhiva.

4.5.10.1. Vrijeme izmjene datoteke

Većina arhivskih formata prema zadanim postavkama bilježi vrijeme zadnje izmjene datoteke, dok će neki također bilježiti vrijeme stvaranja datoteke.
Tar ima način da odredi vrijeme izmjene koje se koristi za sve članove arhive:

```
$ tar --mtime='2015-10-21 00:00Z' -cf product.tar build
```

Slika 18: tar

Za druge arhivske formate uvijek je moguće upotrijebiti touch za ponovno postavljanje vremena izmjene na unaprijed definiranu vrijednost prije stvaranja arhive:

```
$ find build -print0 |
    xargs -0r touch --no-dereference --date="@${SOURCE_DATE_EPOCH}"
$ zip -r product.zip build
```

Slika 19: touch

U nekim slučajevima, poželjno je zadržati izvorna vremena za datoteke koje nisu stvorene ili izmijenjene tijekom procesa izgradnje:

```
$ find build -newermt "@${SOURCE_DATE_EPOCH}" -print0 |
    xargs -0r touch --no-dereference --date="@${SOURCE_DATE_EPOCH}"
$ zip -r product.zip build
```

Slika 20: izvorna vremena

Zakrpa je napisana da pojednostavi potonju operaciju s GNU Tar. Trenutno je dostupan u Debianu od tar verzije 1.28-1. Uskoro bi trebala biti integrirana uzvodno, ali bi se trebalo koristiti s oprezom. Dodaje novu zastavu --clamp-mtime koja će postaviti samo vrijeme kada je datoteka novija od

vrijednosti navedene sa --mtime:

```
# Only in Debian unstable for now
$ tar --mtime='2015-10-21 00:00Z' --clamp-mtime -cf product.tar build
```

Slika 21: clamp

Ovo ima prednost ostavljajući originalno vrijeme izmjene datoteke netaknutim.

4.5.10.2. Redoslijed datoteka

Kada se od njih zatraži da snime direktorije, većina arhivskih formata će čitati njihov sadržaj redoslijedom koji vraća datotečni sustav koji će vjerojatno biti drugačiji pri svakom pokretanju.
S verzijom 1.28, GNU Tar je dobio opciju --sort=name koja će razvrstati nazive datoteka na način neovisan o lokalitetu:

```
# Works with GNU Tar 1.28
$ tar --sort=name -cf product.tar build
```

Slika 22: tar sort

Za starije verzije ili druge arhivske formate, moguće je koristiti find i sort za postizanje istog učinka:

```
$ find build -print0 | LC_ALL=C sort -z |
    tar --no-recursion --null -T - -cf product.tar
```

Slika 23: find i sort

Mora se paziti da se sortiranje poziva u kontekstu C lokaliteta kako bi se izbjegla bilo kakva iznenađenja povezana s redoslijedom uspoređivanja.

4.5.10.3. Korisnici, grupe i numerički ID-ovi

Ovisno o formatu arhive, može se zabilježiti korisnik i grupa koja posjeduje datoteku. Ponekad će koristiti niz, ponekad koristiti pridružene numeričke ID-ove.
Kada datoteke pripadaju unaprijed definiranim grupama sustava, to nije problem, ali se nadogradnje često izvode s običnim korisnicima. Snimanje naziva računa ili njegovih povezanih ID-ova može biti izvor problema s reproducibilnošću.
Tar nudi način da odredite korisnika i grupu koja posjeduje datoteku. Korištenje 0/0 i --numeric-owner je sigurniji pristup, jer će učinkovito zabilježiti 0 kao vrijednosti:

```
$ tar --owner=0 --group=0 --numeric-owner -cf product.tar build
```

Slika 24: vlasništvo

4.5.10.4. PAX zaglavlja

GNU tar zadano postavlja format pax i ako je postavljena POSIXLY_CORRECT varijabla, to dodaje ctime, atime i PID tar procesa kao nedeterminističke metapodatke [38].
Da bi se to izbjeglo, potrebno je poništiti POSIXLY_CORRECT (radi samo s tar>1.32) ili dodati tar pozivu --pax-option=exthdr.name=%d/PaxHeaders/%f,delete=atime,delete=ctime ili --format =gnu (oba dostupna samo u GNU tar) ili koristite --format=ustar ako ograničenja u tom formatu ne predstavljaju problem.

Preporučeni način za stvaranje Tar arhive je sljedeći:

```
# requires GNU Tar 1.28+
$ tar --sort=name \
      --mtime="@${SOURCE_DATE_EPOCH}" \
      --owner=0 --group=0 --numeric-owner \
      --pax-option=exthdr.name=%d/PaxHeaders/%f,delete=atime,delete=ctime \
      -cf product.tar build
```

Slika 25: stvaranje tar arhive

4.5.10.5. Naknadna obrada

Ako alati ne podržavaju opcije za stvaranje reproducibilnih arhiva, uvijek je moguće izvršiti naknadnu obradu.
Strip-nondeterminism već ima podršku za normalizaciju arhiva Zip i Jar (s ograničenjima). Prilagođene skripte poput re-dzip.sh preglednika Tor također mogu biti opcija.

4.5.10.6. Statičke biblioteke

Statičke biblioteke (.a) na sustavima sličnim Unixu su ar arhive. Kao i drugi arhivski formati, one sadrže metapodatke, odnosno vremenske oznake, UID-ove, GID-ove i dopuštenja. Nijedan zapravo nije potreban za njihovo korištenje kao biblioteka.
GNU ar i drugi alati iz binutilsa imaju deterministički način rada koji će koristiti nulu za UID-ove, GID-ove, vremenske oznake i koristiti dosljedne načine rada datoteka za sve datoteke. Može se postaviti kao zadano dodavanjem opcije --enable-deterministic-archives u ./configure. Već je omogućeno prema zadanim postavkama za neke distribucije i do sada se čini prilično sigurnim osim

za Makefile koji koriste ciljeve poput archive.a(foo.o).
Kada binutils nije izgrađen s deterministčkim arhivama prema zadanim postavkama, sustavi izgradnje moraju se promijeniti kako bi se proslijedile prave opcije ar-u. ARFLAGS se može postaviti na Dcvr s mnogim sustavima izgradnje za uključivanje deterministčkog načina rada. Također se mora voditi računa o prosljeđivanju -D ako se ranlib koristi za stvaranje indeksa funkcije.
Druga mogućnost je naknadna obrada s strip-nondeterminism ili objcopy:

```
objcopy --enable-deterministic-archives libfoo.a
```

Slika 26: objcopy

4.5.10.7. Initramfs slike

Cpio arhive se obično koriste za initramfs slike, koje se izgrađuju kod instalacije jezgre i koriste se kod pokretanja operacijskog sustava. Format cpio zaglavlja može sadržavati brojeve uređaja i informacijskog čvora (inode), koji iako su deterministički, mogu varirati od sustava do sustava. Jedan od načina da ih filtrirate je provođenje kroz bsdtar.

```
echo ucode.bin |
    bsdcpio -o -H newc -R 0:0 > ucode.img
```

Slika 27: Primjer nedeterminističkog koda

```
echo ucode.bin |
    bsdtar --uid 0 --gid 0 -cnf - -T - |
    bsdtar --null -cf - --format=newc @- > ucode.img
```

Slika 28: Primjer deterministčkog koda

Imajte na umu da druge poteškoće poput vremenskih oznaka još uvijek mogu zahtijevati ispravljanje prije arhiviranja.

4.5.10.8. GNU Libtool

GNU Libtool prije Git commita s ID-em 74c8993c (prvi put uključen u verziju 2.2.7b) nije sortirao izlaz traženja. Čini se da se mnogi paketi podižu s verzijom koja je prethodila ovoj.
Zbunjujuće, iako GNU GCC-ov ltmain.sh tvrdi da ga je generirao libtool 2.2.7a, GNU GCC zapravo održava vlastitu verziju libtool.m4 i ltmain.sh, koji su neovisno riješili ovaj problem u Git

commitu s ID-em d41cd173e23. Ova gore navedena promjena prvi put je uključena u verziju 9.1.0, što znači da problem reproducibilnosti ostaje u GCC verzijama ispod te.

4.5.11. Stabilan poredak za izlaze

Podatkovne strukture kao što su Perl hashovi, Python riječnici ili Ruby Hash objekti navest će svoje ključeve različitim redoslijedom pri svakom pokretanju kako bi se ograničili napadi na složenost algoritama [39].
Sljedeći Perl kod će ispisati popis različitim redoslijedom pri svakom pokretanju:

```
foreach my $package (keys %deps) {
    print MANIFEST, "$package: $deps[$packages]";
}
```

Slika 29: Perl kod

Da biste dobili deterministički izlaz, najlakši način je eksplicitno sortirati ključeve:

```
foreach my $package (sort keys %deps) {
    print MANIFEST, "$package: $deps[$packages]";
}
```

Slika 30: eksplicitno sortiranje ključeva

Za Perl je moguće postaviti PERL_HASH_SEED=0 u okruženju. To će rezultirati time da hash ključevi uvijek budu u istom redoslijedu.
Korisnici Pythona mogu na sličan način postaviti varijablu okruženja PYTHONHASHSEED. Kada se postavi na zadanu vrijednost cijelog broja, poredak u rječnicima bit će isti pri svakom pokretanju. Pazite na to da lokalne postavke mogu utjecati na izlaz nekih funkcija sortiranja ili naredbe sort.

4.5.12. Nasumičnost

Nasumični podaci učinit će konstrukcije nereproducibilnima i moraju se izbjegavati.
Ako je potreban unos nalik nasumičnom, rješenje je korištenje unaprijed određene vrijednosti za postavljanje generatora pseudoslučajnih brojeva. Ova se vrijednost može pročitati iz neke datoteke, dnevnika promjena ili sustava za kontrolu verzija.
Kada su uključene optimizacije vremena povezivanja, GCC korisnici će pisati nasumične identifikatore u binarne objekte koje kreiraju. Korištenje -frandom-seed može se koristiti za ovaj

poseban slučaj. Budući da će hashirati proizvoljne podatke, prosljeđivanje naziva datoteke trebalo bi funkcionirati u većini slučajeva.
Neki alati za kompilaciju će napisati međuprivremene datoteke. To može dovesti do problema s reproducibilnošću ako se putanje ugrade u konačni izlaz. Nema općih rješenja za takve slučajeve, bolje je izravno prepraviti kôd. Jedan od načina je korištenje .file asemblerske direktive.

4.5.13. Putanje izgradnje

Neki alati će zabilježiti put izvornih datoteka u svom izlazu. Većina kompajlera piše putanju izvora u informacijama za ispravljanje pogrešaka kako bi locirali pridružene izvorne datoteke.
Neki alati imaju zastavice (poput gzipove -n) koje ih sprječavaju da zapišu putanju u svom izlazu. Predlaganje zakrpa za dodavanje slične značajke drugim alatima moglo bi biti dovoljno jednostavno. Međutim, u većini slučajeva potrebna je naknadna obrada kako bi se uklonila staza izgradnje ili normalizirala na unaprijed definiranu vrijednost.

Za specifičan slučaj simbola za otklanjanje pogrešaka, trenutno ne postoji dobar alat za naknadnu obradu za njihovu promjenu na unaprijed određenu vrijednost. Zaobilazno rješenje je definiranje putanje izgradnje kao dijela okruženja izgradnje, no ponovni protest to mijenja, pa je teže procijeniti reproducibilnost. Određene zastavice kompajlera mogu zaobići problem [40]:

- -fdebug-prefix-map=OLD=NEW može ukloniti prefikse direktorija iz informacija o otklanjanju pogrešaka. (dostupno u svim GCC verzijama, Clang 3.8)
- -fmacro-prefix-map=OLD=NEW sličan je -fdebug-prefix-map, ali se bavi nereproducibilnošću zbog upotrebe makronaredbi __FILE__ u assert pozivima. (dostupno od GCC 8 i Clang 10)
- -ffile-prefix-map=OLD=NEW je alias za -fdebug-prefix-map i -fmacro-prefix-map. (dostupno od GCC 8 i Clang 10)

S dpkg >= 1.19.1, prvi put isporučenim s Debian Busterom, paketi mogu omogućiti zastavu -ffile-prefix-map=OLD=NEW dodavanjem zastave za izgradnju fixfilepath u njihovu datoteku debian/rules. Primjerice:

```
export DEB_BUILD_MAINT_OPTIONS = hardening=+all reproducible=+fixfilepath
```

Slika 31: fixfilepath

Imajte na umu da neki paketi spremaju opcije kompajliranja u izlazu izgradnje. Ovo je također

problematično jer će se primijeniti na srednje izvorne datoteke koje generiraju drugi alati. Budući da obično koriste nasumične nazive datoteka, u takvim slučajevima nije dovoljna fiksna putanja izgradnje.

4.5.14. Slike datotečnog sustava

Opći problemi reproducibilnih slika datotečnog sustava su idući [41]:

- Datotečni sustav treba stvoriti odjednom
- Datotečni sustavi imaju vremenske oznake stvaranja i/ili izmjene
- Datotečni sustavi sadrže UUID ili oznaku koji nisu eksplicitno postavljeni
- Uključene datoteke imaju vremenske oznake
- Uključene datoteke mogu se generirati ili ažurirati tijekom izgradnje na način koji se nije reproducibilan
- Bootloader koji je integriran može imati vremenske oznake
- Integrirane initramfs slike mogu se izgraditi neponovljivo tijekom izgradnje slike sustava
- Vremenske oznake koje ne ovise o SOURCE_DATE_EPOCH nisu reproducibilne

4.5.14.1. Datotečni sustav u formatu ISO

Prilikom izgradnje ISO datotečnih sustava s xorriso:

- koristite najnovije verzije xorrisa koji poštuju $SOURCE_DATE_EPOCH za razne metapodatke ISO slike
- proslijedite $SOURCE_DATE_EPOCH xorrisovom --modification-date da približite sva vremena.

Također bi moglo biti potrebno:

- proslijedite fiksnu vrijednost u isohybrid –i
- osigurate da su initrd slike izgrađene reproducibilno

4.5.14.2. SquashFS metapodaci i kompresija

Prilikom komprimiranja SquashFS slika, metapodaci i kompresija mogu učiniti izlaz

nereproducibilnim. Prilikom izrade SquashFS slika, starije verzije alata ponekad su davale nereproducibilne rezultate. Dobar mksquashfs će [41]:

- Poštovati $SOURCE_DATE_EPOCH za razne vremenske oznake
- Sažeti vremenske oznake sadržaja na $SOURCE_DATE_EPOCH
- Ne promjeniti redoslijed fragmenata na temelju uvjeta višenitnosti

4.5.15. Sadržaj korijenskog datotečnog sustava

Slika sustava često sadrži korijenski datotečni sustav, generiran tijekom izgradnje i upakiran u neki format kao što je SquashFS [41].

4.5.15.1. Izuzimanje nepotrebnih datoteka

Određeni broj datoteka može se jednostavno isprazniti ili isključiti prilikom stvaranja slike korijenskog datotečnog sustava (neke za optimizaciju veličine, neke jer nisu potrebne), što čini Tails. Ovo treba raditi s oprezom, pošto može imati teško predvidljive posljedice. Primjerice, Tails je razmatrao o izbacivanju predmemorija fontconfig -, ali su primjećeni neočekivani rezultati i problemi s performansama pri tome i konačno je ideaj odbačena.

4.5.15.2. Metapodaci datoteka

Proces izgradnje za sliku sustava često stvara ili ažurira datoteke, što generira metapodatke datoteke koji ovise o tome kada se izgradnja izvodi.
Jedan pristup koji je uspješno korišten za rješavanje ovog problema je stezanje mtimes vremena datoteka u korijenskom datotečnom sustavu na $SOURCE_DATE_EPOCH prije generiranja slike sustava.

4.5.15.3. Datoteke generirane ili ažurirane tijekom izgradnje

Upravitelji paketa kao što su dpkg ili RPM podržavaju postinst skripte i okidače koji se pokreću na ciljnom sustavu nakon raspakiranja paketa, npr. za generiranje ili ažuriranje predmemorija i indeksa, potencijalno na nedeterministički način.
Kako bi se tome suprotstavili, jedan mogući pristup je zamijeniti te skripte.
Drugi pristup je osigurati da te skripte generiraju/ažuriraju datoteke na način koji je reproducibilan. Ovaj pristup ima prednost u rješavanju temeljnog uzroka problema i njegovom popravljanju za svaki projekt koji koristi te programe.

4.5.15.4. gettext

GNU-ove gettext POT, PO i MO datoteke predstavljaju zanimljiv izazov. Jedan od načina za pristup ovom problemu je [41]:

- ažurirajte POT datoteke samo kada je to stvarno potrebno, primjerice ako je polje POT-Creation-Date jedina promjena nakon osvježavanja POT datoteke, datoteka se ne mora ažurirati;
- izbjegavajte ažuriranje PO - a time i MO - datoteka kada su promijenjeni samo komentari (npr. brojevi redaka).

4.5.16. Java Virtual Machine (JVM)

JVM ekosustav nudi mnoge jezike i alate za izgradnju. Budući da JVM nije pogodan za reproducibilnost od samog početka – Jar i Zip datoteke su prvi prirodni izvor varijacija, s redoslijedom datoteka i vremenskom oznakom, svaki alat za izradu zahtijeva nešto posla kako bi pružila reproducibilnost.

4.5.16.1. Središnjica za reproducibilnost

Bez obzira na alat za izradu, binarni JVM artefakti općenito se objavljuju u repozitoriju artefakata koji koriste Maven2 format repozitorija (koristeći groupId/artifactId/version koordinate) kao što je Maven Central ili Googleov Android Repository.
Središnjica za reproducibilnost pokušaj je ponovne izgradnje javnih izdanja objavljenih na Maven Centralu i provjere može li se postići reproducibilna gradnja.

4.5.17. Konfiguriranje alata za izradu za reproducibilnost

4.5.17.1. Maven

Dobivanje reproducibilnih izgradnji s Mavenom zahtijeva konfiguraciju dodataka.

Za početak, potrebno je nadograditi dodatke na reproducibilne verzije: da bi se otkrile potrebne nadogradnje, pokrenite:

```
mvn artifact:check-buildplan
```

Slika 32: maven

Omogućite način rada

Reproducible Builds za dodatke dodavanjem svojstva project.build.outputTimestamp u pom.xml projekta[42]:

```
<properties>
  <project.build.outputTimestamp>10</project.build.outputTimestamp>
</properties>
```

Slika 33: maven 2

4.5.17.2. Gradle

Gradle podržava reproducibilne arhive od v3.4. Zadaci koji generiraju arhive, kao što su Zip ili Jar, mogu nametnuti sačuvane vremenske oznake datoteka i ponovljivi redoslijed datoteka koji popravljaju dva glavna izvora nedeterminizma u JVM artefaktima.

4.5.17.3. sbt

Korištenjem sbt-a, alata za izgradnju popularnom u Scala projektima, može se koristiti dodatak sbt-reproducible-builds za uklanjanje artefakata i dijeljenje informacija o izgradnji.

4.6. Definiranje okruženja za igradnju

Reproducibilne izgradnje ne zahtijevaju da se određeni dio izvornog koda pretvori u iste bajtove u svim situacijama. To bi bilo neizvedivo. Izlaz kompajlera vjerojatno će se razlikovati od jedne verzije do druge jer se sve bolje optimizacije integriraju cijelo vrijeme.
Umjesto toga, ponovljive izgradnje događaju se u kontekstu građevne okoline. Obično se sastoje od skupa alata, potrebnih verzija i drugih pretpostavki o operativnom sustavu i njegovoj konfiguraciji. Opis ovog okruženja obično treba biti zabilježen i dat uz svaki distribuirani binarni paket.

4.6.1. Zahtjevi

Ono što točno čini okruženje za izgradnju bit će različito za svaki projekt. Može postojati nekoliko okruženja za izgradnju za jedno izdanje kako bi se prilagodili različitim ciljnim operativnim sustavima. Ali postoje neki važni aspekti zajednički svim sredinama.
Trebalo bi biti lako instalirati odgovarajuće okruženje za izgradnju na sustave korisnika. Idealno bi bilo da se sastoji samo od besplatnog softvera dostupnog na javnim internetskim stranicama.
Najbolji način za pružanje okruženja je korištenje dokumentirane i lako razumljive skripte.
Trebao bi se moći revidirati. Mora biti lako razumjeti koji su alati dio okruženja za izgradnju. Te bi

bilo idealno da ih je lako pregledati i ponovno izgraditi.

4.6.2. Sadržaj

Opseg okruženja izgradnje mora biti pravilno specificiran jer će to odrediti koja količina sustava izgradnje treba biti deterministička. Definirano okruženje za izgradnju ima popis alata koje koristi proces izgradnje i njihove verzije. Ostatak se može razlikovati od jednog projekta do drugog, sve dok ga mogu reproducirati zainteresirani korisnici. Primjerice [43]:

- specifični operativni sustav (ako unakrsna kompilacija nije podržana),
- arhitektura sustava izgradnje (ako unakrsno prevođenje nije podržano),
- direktorij gdje se mora dogoditi izgradnja,
- ime korisnika koji pokreće izgradnju,
- lokalizacije,
- Vremenska zona,
- specifične varijable okruženja (kao što je SOURCE_DATE_EPOCH).

Korištenje virtualnih strojeva ili spremnika kao preporučenog okruženja za izgradnju može olakšati osiguravanje specifičnog operacijskog sustava ili korisničke konfiguracije. Ali oni također mogu sakriti neke pretpostavke o okruženju, kao što su određene optimizacije omogućene zbog vrste procesora sustava.

4.6.3. Snimanje okoline izgradnje

Uobičajeno je u softveru usmjerenom prema korisniku da se programerima koji istražuju pogreške omogući način da saznaju kako je softver napravljen [44]. “About dialog” ili izlaz --version-a obično sadrži informacije o okruženju za izradu.
U kontekstu reproducibilnih gradnji, ili aktivno činimo aspekte okoline gradnje nevažnima za izlaz gradnje, ili osiguravamo da su dostupni za ponovnu izgradnju softvera točno onako kako su distribuirani.
Sve relevantne informacije o okruženju izgradnje treba definirati kao dio procesa razvoja ili zabilježiti tijekom procesa izgradnje.

4.6.4. Format datoteke

Sve što je snimljeno najbolje je pohraniti kao zaseban proizvod koji se lako može zanemariti ili distribuirati zasebno. To će pomoći u prepoznavanju koja varijacija nije relevantna za sam softver [44].
Ovaj se proizvod naziva .buildinfo, ali njegov točan format i način na koji se distribuira razlikuju se među ekosustavima.

4.6.4.1. Debian

Debian dijeli svoje datoteke .buildinfo kao obične tekstualne datoteke slijedeći format kontrolne datoteke, obično s jasnim potpisom s OpenPGP-om.

4.6.4.2. Arch Linux

Arch Linux makepkg alat za izgradnju proizvodi datoteku .buildinfo koja se sastoji od parova <key> = <value> [45]. Za razliku od Debiana, ova datoteka nije samostalno potpisana i distribuirana, već je uključena u paket (i stoga potpisana kao dio potpisa paketa).

4.6.4.3. Tails

Tails ne bilježi datoteku .buildinfo direktno, ali umjesto toga „lutajući” direktorij glavnog git repozitorija sadrži sve informacije potrebne za reproducibilnu ponovnu izgradnju te revizije Tailsa.

4.6.4.4. JVM

U JVM ekosustavu uobičajeno je distribuirati biblioteke kao binarne (bajt-kod) jar datoteke prenesene u repozitorij kao što je Maven Central ili Googleov Android Repository.
Preporuča se da se uz svaki artefakt objave informacije o izgradnji koje opisuju okruženje izgradnje koje se koristi za tu službenu izgradnju. Potvrde treće strane mogu se dijeliti u zasebnom sig repozitoriju.

4.6.5. Strategije definiranja

Postoji više načina za definiranje okruženja za izgradnju na način da se može distribuirati. Sljedeće metode nisu isključivo jedine i više se aspekata može koristiti za jedan projekt [46].
Definiranje okruženja za izgradnju kao dijela razvojnog procesa ima vrlo poželjan aspekt: promjene u okruženju za izgradnju mogu se provjeriti kao i sve druge promjene. Ažuriranje na novu verziju

prevoditelja može biti podložno recenziji, automatskom testiranju i u slučaju kvara vraćanju na staru verziju.

4.6.5.1. Gradnja iz izvora

Jedan način da korisnici reproduciraju alate koji se koriste za izradu je natjerati ih da počnu graditi pravu verziju ovih alata iz izvora. Korištenjem make ili bilo kojeg drugog upravljačkog programa za kompilaciju, potrebni alati će se preuzeti, izgraditi i lokalno instalirati prije kompajliranja softvera. Kao i svaki drugi ulaz s mreže, sadržaj arhive u kojoj je pohranjen traženi izvor za alate treba sigurnosno kopirati i provjeriti pomoću kriptografskih kontrolnih zbrojeva.

4.6.5.2. Referentna distribucija

Korištenje određene verzije distribucije besplatnog softvera još je jedna održiva opcija za okruženje za izgradnju. U idealnom slučaju, trebao bi ponuditi stabilna izdanja (poput Debiana, CentOS-a ili FreeBSD-a) kako bi se izbjeglo stalno ažuriranje dokumentacije ili skripti za izgradnju. Bilježenje točnih verzija instaliranih paketa može biti korisno za dijagnosticiranje problema. Neke distribucije također čuvaju potpunu povijest izvornih paketa ili binarnih paketa dostupnih za kasniju instalaciju.

4.6.5.3. Virtualni strojevi i spremnici

Neki aspekti okruženja za izgradnju mogu se prilično pojednostaviti korištenjem virtualnih strojeva ili spremnika. S virtualnim strojem možete jednostavno izvesti izgradnju u kontroliranijem okruženju. Korisnik izgradnje, naziv hosta sustava, mrežna konfiguracija ili drugi aspekti mogu se jednostavno nametnuti na svim sustavima.
Loša strana je što može uvesti puno softvera kojem se nekako mora vjerovati. Primjerice, trenutno nije moguće instalirati Debian na reproducibilan način. To otežava usporedbu različitih instalacija.

4.6.6. Vlasnički operacijski sustav

Na vlasničkim operacijskim sustavima teško je utvrditi jesu li neovlašteno mijenjani. Obično zahtijevaju i komercijalne alate za kompilaciju koje je korisnicima teško nabaviti.
Dobra vijest je da za neke slučajeve postoje besplatni softverski alati koji mogu unakrsno prevesti softver za vlasničke operativne sustave na besplatnim operativnim sustavima; i Bitcoin i Tor Browser su bili pioniri u tehnici za izgradnju svojih Windows i Mac OS X verzija korištenjem unakrsnog prevođenja.

4.6.6.1. Windows

Za Windows, mingw-w64 će izgraditi Windows binarne datoteke na POSIX kompatibilnim operativnim sustavima [47]; NSIS se može koristiti za stvaranje integriranog instalacijskog paketa. Oba su dostupna u nekoliko besplatnih distribucija softvera.

4.6.6.2. MAC OS X

Crosstool-ng bi trebao raditi za izradu softvera za Mac OS X. Nažalost, čini se da ovo zahtijeva dio Apple SDK-a koji se ne može redistribuirati. Može se izdvojiti iz XCodea koji se može besplatno preuzeti [48].
Softver iz Mac OS X često se distribuira kao slike diska (.dmg) koje se mogu izraditi pod GNU/Linuxom, ali trenutno zahtijeva više različitih alata [49].

4.6.7. Izgradnja iz izvora

Izgradnja alata koji stvaraju okruženje iz izvora jedan je od načina da se korisniku omogući njegova reprodukcija. Izravno korištenje izvornog koda olakšava oslanjanje na nove značajke i jednostavno funkcionira na različitim platformama [50]. Možda se neće dobro mjeriti za dugačak popis zavisnosti, a traženje od korisnika da ponovno izgrade GCC za svaki dio softvera koji koriste moglo bi ih učiniti pomalo nezadovoljnima.
Ono što slijedi su prijedlozi o tome kako se nositi s izgradnjom alata za kompilaciju iz izvora.

4.6.7.1. Izgradnja korištenjem vanjskih resursa

Izvor za različite komponente može se dohvatiti iz mrežnih repozitorija. Korištenje tarball izdanja moglo bi biti bolje jer ih je lakše predmemorirati, zrcaliti, provjeriti kontrolni zbroj i verificirati. Kada dohvaćate izvor iz repozitorija sustava za kontrolu verzija, najbolje je imati preciznu referencu na verziju koda. S Gitom će najbolje funkcionirati korištenje oznake s potvrđenim potpisom ili commit hashom.
Samu kompilaciju mogu pokretati skripte ljuske ili dodatni cilj u projektu Makefile.
coreboot je dobar primjer. Dokumentacija izgradnje nalaže prvo pokretanje make crossgcc prije izgradnje samog coreboota.

4.6.7.2. Provjera izvora lanca alata za izgradnju

Drugi pristup je provjeriti izvor cijelog lanca alata u sustavu kontrole verzija projekta.

Tako je razvijeno nekoliko integriranih operativnih sustava poput BSD-a. "Izgradnja svijeta" započet će izgradnjom lanca alata u sustavu kontrole verzija prije izgradnje ostatka sustava. Googleovi interni projekti također funkcioniraju na ovaj način. Izdali su Bazel koji se temelji na njihovom internom alatu za kompilaciju. Bazel je dizajniran za pokretanje velikih konstrukcija imajući na umu brzinu i reproducibilnost.

4.6.7.3. Slanje lanca alata za izgradnju kao izgrađenog proizvoda

Budući da bi moglo biti teško tražiti od svakog korisnika da potroši vrijeme na ponovnu izgradnju cijelog lanca alata, OpenWrt daje dobar primjer srednjeg tla. "SDK" koji se može preuzeti zajedno sa slikama sustava koji sadrži sve što je potrebno za izgradnju ili ponovnu izgradnju dodatnih paketa. U tom slučaju SDK postaje još jedan proizvod za izgradnju i mora biti moguće izgraditi ga reproducibilno.

4.6.8. Pomoćni programi za virtualne strojeve i kontejnere

Ako je okruženje za izgradnju definirano korištenjem operativnog sustava bez referenci, prisiljavanje korisnika da ponovno instaliraju cijelo računalo samo da bi reproducirali izgradnju predstavlja veliki zahtjev.
Danas imamo virtualne strojeve i kontejnere (npr. LXC ili Docker kontejnere) koji mogu omogućiti jednostavan pristup širokom rasponu operativnih sustava uz male troškove. Dizajnirano je nekoliko alata za pomoć pri pokretanju virtualnih strojeva i spremnika, neki s izričitim ciljem reproducibilnih nadogradnji [51].

4.6.8.1. Gitian

Gitian je alat koji koristi Bitcoin [52]. Radi ili u Linux spremniku pomoću LXC-a ili u virtualnom računalu pomoću KVM-a.
Gitian uzima "deskriptore" kao ulaz koji govore koju osnovnu GNU/Linux distribuciju koristiti, koje pakete instalirati, koje Git daljinske upravljače treba dohvatiti, koje druge ulazne datoteke koristiti i skriptu za izgradnju koju treba pokrenuti sa svim tim. Kao što je ranije objašnjeno, korištenje virtualnog stroja pomaže da se riješite nekoliko dodatnih varijacija koje se mogu dogoditi od jednog sustava do drugog.

4.6.8.2. rbm

U procesu izrade preglednika Tor, rbm je alat odgovoran za određivanje komponenti koje je

potrebno (ponovno) izgraditi, preuzimanje ovisnosti, provjeru njihovih potpisa ili kontrolnih zbrojeva, stvaranje slika spremnika s odabranim paketima i pokretanje izgradnje svake komponente unutar spremnika koristeći runc [53] [54].

4.6.8.3. Docker

Olakšavanje postavljanja i korištenja spremnika problem je koji Docker pokušava riješiti [55]. Docker datoteke se koriste za opis kako stvoriti spremnik i kako se aplikacije mogu pokrenuti u njemu.
Alat koji se često koristi u Docker slikama, gosu se može reproducibilno izgraditi pomoću Dockera. Koristeći referentni spremnik koji je dostupan za izradu Go aplikacija, zatim instalira potrebne ovisnosti i poziva Go kompajler u tom okruženju koje bi trebalo biti uglavnom isto cijelo vrijeme. Kako bismo bili sigurni da je osnovni kompajler isti, mogli bismo iskoristiti činjenicu da se Docker slikama zapravo može adresirati hashom njihovog sadržaja. Druga je mogućnost da sami napravite Docker sliku na način koji se može reproducirati, nešto što se može učiniti pomoću Bazela.

4.6.8.4. Vagrant

Vagrant je još jedan alat, napisan u Rubyju, koji može pokretati virtualne strojeve korištenjem VirtualBoxa [56]. Također se može koristiti za dobivanje kontroliranog okruženja za izgradnju. Dobra strana Vagranta i VirtualBoxa je da rade na Mac OS X i Windows, pa bi to moglo pomoći većem broju korisnika da stvarno provjere je li na građenje utjecao neki vanjski čimbenik.

4.6.9. Formalna definicija

Većina distribucija besplatnog softvera je samostalna: svi alati potrebni za izgradnju njihovih komponenti dio su distribucije. U takvim slučajevima moguće je navesti okruženje za izradu u strojno čitljivom formatu koji se kasnije može koristiti za ponovnu instalaciju okruženja. Primjerice, .buildinfo kontrolne datoteke koje koristi Debian povezuju u istu datoteku: izvore, generirane binarne datoteke i sve pakete korištene za izvođenje izgradnje (s točnim brojem verzije).

4.7. Alati

Dostupno je nekoliko alata koje vrijedi spomenuti jer olakšavaju rad s reproducibilnim izgradnjama softvera.

4.7.1. Diffoscope

diffoscope će pokušati doći do dna onoga što datoteke ili direktorije čini različitima [57]. Rekurzivno će raspakirati arhive mnogih vrsta i transformirati različite binarne formate u čitljivije oblike za usporedbu. Jednako lako može usporediti dvije tarball datoteke, ISO slike ili PDF-ove.

4.7.2. Rebuilderd

rebuilderd nadzire repozitorij paketa Linux distribucije i koristi pozadinske programe rebuildera kao što je archlinux-repro za provjeru da se dani binarni paketi mogu reproducirati iz objavljenog izvornog koda [58].

4.7.3. Archlinux-repro

repro je backend program koji provjerava dani Arch Linux paket [59]. Koristi ugrađenu .BUILDINFO datoteku za rekonstrukciju identičnog okruženja za izgradnju i ponavlja izgradnju iz izvora, zatim uspoređuje ulazni paket s paketom generiranim tijekom verifikacije izgradnje.

4.7.4. Disorderfs

Ponekad je teško pronaći probleme s nestabilnim redoslijedom unosa ili drugim varijacijama koje uvode datotečni sustavi. disorderfs je preklapajući FUSE datotečni sustav koji namjerno uvodi nedeterminizam u metapodatke datotečnog sustava [60]. Primjerice, može nasumično odrediti redoslijed čitanja unosa imenika.

5. Verifikacija

5.1. Kriptografske kontrolne sume

Kako korisnici mogu znati da je gradnja koju su upravo napravili uspješno reproducirala izvornu verziju?
Najlakši način je osigurati da su izlazi izgradnje uvijek identični bajt po bajt. Usporedba bajt za bajt je trivijalna operacija i može se izvesti u mnogo različitih okruženja.
Druga prednost postojanja identičnih bajtova je ta što omogućuje korištenje kriptografskih kontrolnih zbrojeva. Takve su kontrolne sume stvarno malene u usporedbi s potpunim proizvodima. Lako se izmjenjuju čak i u situaciji s vrlo malom propusnošću.
Primjerice, omogućuje izradu izdanja softvera i na dobro povezanom (ali teško pouzdanom) poslužitelju i na prijenosnom računalu iza loše mobilne veze. Digitalni potpis moguće je napraviti lokalno na prijenosnom računalu. Kako će proizvodi za izgradnju biti identični, potpis će biti valjan za datoteke proizvedene na dobro povezanom poslužitelju.

5.2. Ugrađeni potpisi

Softver koji se distribuira pomoću ugrađenih kriptografskih potpisa može predstavljati izazov za dopuštanje korisnicima reproduciranja identičnih rezultata [61]. Po definiciji, oni neće moći generirati identičan potpis. To se može riješiti tako da potpis postane dio ulaza procesa izgradnje ili ponudom alata za transformaciju distribuiranih binarnih datoteka u netaknute rezultate izgradnje.

5.2.1. Lijepljenje potpisa

Jedan od načina za rukovanje ugrađenim kriptografskim potpisima je da potpis postane (neobavezni) ulaz procesa izgradnje. Kada je potpis dostupan, samo se kopira na pravo mjesto.
To omogućuje sljedeći tijek rada [61]:

1. Početnu izgradnju izrađuju programeri koji imaju pristup privatnom ključu.
2. Rezultat izgradnje je potpisan u vanjsku datoteku.
3. Potpis je dio objavljenog izvornog koda.
4. Gradnja koja će se distribuirati napravljena je iz drugog izvora.

Wireless-regdb paket u Debianu je primjer kako se to može implementirati.

5.2.2. Ignoriranje potpisa

Može se staviti na raspolaganje određeni alat za usporedbu koji se može uspoređivati s verzijama koje preskaču potpise. U idealnom slučaju, trebao bi također moći proizvesti kriptografske kontrolne zbrojeve kako bi preuzimanje izvorne verzije bilo nepotrebno samo za usporedbu rezultata.
Takav alat mora biti vrlo jednostavan za reviziju i razumijevanje. U suprotnom, teško je vjerovati da skripta ne zanemaruje bajtove zbog kojih bi se ponašala drugačije.

5.2.3. Odstranjivanje potpisa

Druga je mogućnost isporuka alata koji može ukloniti potpise iz službenih izdanja [61]. Rezultat se tada može usporediti bajt po bajt s rezultatima korisnika.
Loša strana ove metode je da korisnik mora preuzeti službena izdanja za usporedbu. Također je teže potvrditi da podaci koji se uklanjaju neće učiniti da se softver ponaša drugačije.

5.3. Dijeljenje certifikacija

Nameće se pitanje kako bi korisnici mogli steći povjerenje da izrada nije ugrožena razmjenom certifikata koji potvrđuju da su svi uspjeli dobiti iste rezultate izrade. Kao odgovor na to, Debian razmišlja o tome da omogući višestrukim Debian programerima učitavanje potpisa koji potvrđuju da su uspjeli reproducirati izgradnju [62].
Pitanje je također povezano s radom koji vodi Ben Laurie o binarnoj transparentnosti. Ideja je imati dnevnik samo za dodavanje sličan Certificate Transparency koji bi se mogao koristiti za provjeru autentičnosti binarnih datoteka.
Potrebno je više istraživanja u ovom području kako bi se reproducibilne izgradnje učinile učinkovitijima u ranom otkrivanju kompromisa.

6. Zaključak

Izraz "reproducibilne konstrukcije" često je preopterećen kontekstom, ali istinski reproducibilne konstrukcije su one koje, s obzirom na iste ulaze, mogu proizvesti identične binarne artefakte bez obzira na originalni stroj izrade ili kada je dani artefakt stvoren. Oni su ključni dio osiguranja lanca nabavke softvera i pomažu osigurati da dobavljači i korisnici softvera znaju točno što se isporučuje. To omogućuje da se brzo utvrde ranjive komponente i isprave kako bi se izbjeglo postojanje ranjivosti i mogućnost njihove zlouporabe.

Za projekte otvorenog koda, dopuštaju korisnicima provjeru odgovaraju li izgrađeni artefakti izvornom kodu u repozitoriju. Čak i ako nije moguće da određeni softver odmah postigne binarnu reproducibilnu izgradnju, poduzimanje malih koraka prema binarnim reproducibilnim verzijama vrijedno je truda i koristi cijelom ekosustavu razvoja softvera, a pogotovo slobodnog softvera otvorenog koda koji se distribuira u obliku izvornog koda zajedno s binarnim oblikom.

Pripadnici projekta reproducibilnih izgradnji softvera su entuzijasti slobodnog softvera otvorenog koda. Unatoč tome što je proteklih nekoliko godina proteklo u ograničenjima zbog pandemije virusa SARS-CoV-2, projekt kao takav nije jenjavao te je službena dokumentacija bila nadopunjavana na redovnoj bazi. Uz tehničke specifikacije i upute za cijeli proces ponovljive izgradnje i naknadnu verifikaciju čitavog postupka, suradnici na ovom projektu su stvorili pravu zajednicu, zajednicu stvorenu od ljudi raširenih diljem svijeta. Bez obzira na geografsku udaljenost, oni su se uspijevali sastajati kako bi mogli održavati duh pripadnosti projektu te mu time osigurali svjetlu budućnost.

Literatura

[1] ‘Strawhorse: Attacking the MacOS and iOS Software Development Kit’. The Intercept, Mar. 10, 2015. Accessed: Dec. 07, 2022. [Online]. Available: https://theintercept.com/document/2015/03/10/strawhorse-attacking-macos-ios-software-development-kit/

[2] C. Xiao, ‘Novel Malware XcodeGhost Modifies Xcode, Infects Apple iOS Apps and Hits App Store’, *Unit 42*, Sep. 17, 2015. https://unit42.paloaltonetworks.com/novel-malware-xcodeghost-modifies-xcode-infects-apple-ios-apps-and-hits-app-store/ (accessed Dec. 07, 2022).

[3] *Reproducible Builds*, (00:00 100AD). Accessed: Dec. 07, 2022. [Online Video]. Available: https://media.ccc.de/v/31c3_-_6240_-_en_-_saal_g_-_201412271400_-_reproducible_builds_-_mike_perry_-_seth_schoen_-_hans_steiner

[4] ‘#773916 - libical: Ship different constant values accross builds - Debian Bug report logs’. https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=773916 (accessed Dec. 07, 2022).

[5] K. Thompson, ‘Reflections on trusting trust’, *Commun. ACM*, vol. 27, no. 8, pp. 761–763, Aug. 1984, doi: 10.1145/358198.358210.

[6] D. A. Wheeler, ‘Fully Countering Trusting Trust through Diverse Double-Compiling’. arXiv, Apr. 30, 2010. doi: 10.48550/arXiv.1004.5534.

[7] ‘Gitian: a secure software distribution method’. https://gitian.org/ (accessed Nov. 07, 2022).

[8] ‘Global surveillance disclosures (2013–present)’, *Wikipedia*. Dec. 01, 2022. Accessed: Dec. 07, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Global_surveillance_disclosures_(2013%E2%80%93present)&oldid=1124956490

[9] ‘Deterministic Builds Part One: Cyberwar and Global Compromise | Tor Project’. https://blog.torproject.org/deterministic-builds-part-one-cyberwar-and-global-compromise/ (accessed Dec. 07, 2022).

[10] ‘dc13: Byte-for-byte identical reproducible builds?’, Sep. 21, 2014. https://penta.debconf.org/dc13_schedule/events/1063.en.html (accessed Dec. 07, 2022).

[11] ‘ReproducibleBuilds - Debian Wiki’. https://wiki.debian.org/ReproducibleBuilds?action=recall&rev=1 (accessed Dec. 07, 2022).

[12] ‘History — reproducible-builds.org’. https://reproducible-builds.org/docs/history/ (accessed Nov. 07, 2022).

[13] ‘Reproducible Builds for Debian’. https://archive.fosdem.org/2014/schedule/event/reproducibledebian/ (accessed Dec. 07, 2022).

[14] ‘[Reproducible-builds] Lintian checker for gzip files’. https://alioth-lists.debian.net/pipermail/reproducible-builds/Week-of-Mon-20140210/000038.html (accessed Nov. 07, 2022).

[15] ‘[Reproducible-builds] debugedit and reproducible builds’. https://alioth-lists.debian.net/pipermail/reproducible-builds/Week-of-Mon-20140217/000053.html (accessed Dec. 07, 2022).

[16] ‘5. Control files and their fields — Debian Policy Manual v4.6.1.1’. https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-debiansourcecontrolfiles (accessed Dec. 07, 2022).

[17] ‘4. Source packages — Debian Policy Manual v4.6.1.1’. https://www.debian.org/doc/debian-policy/ch-source.html#s-dpkgchangelog (accessed Dec. 07, 2022).

[18] ‘Reproducible Builds / strip-nondeterminism · GitLab’, *GitLab*. https://salsa.debian.org/reproducible-builds/strip-nondeterminism (accessed Nov. 07, 2022).

[19] ‘Debugging Options - Using the GNU Compiler Collection (GCC)’. https://gcc.gnu.org/onlinedocs/gcc-4.9.2/gcc/Debugging-Options.html (accessed Dec. 07, 2022).

[20] ‘[Reproducible-builds] How to help out sorting logs?’ https://alioth-lists.debian.net/pipermail/reproducible-builds/Week-of-Mon-20140217/000065.html (accessed Dec. 07, 2022).
[21] ‘[Reproducible-builds] A recap of DebConf14 for reproducible builds in Debian’. https://alioth-lists.debian.net/pipermail/reproducible-builds/Week-of-Mon-20140901/000198.html (accessed Dec. 07, 2022).
[22] ‘Making plans — reproducible-builds.org’. https://reproducible-builds.org/docs/plans/ (accessed Dec. 07, 2022).
[23] ‘Comparison of cryptographic hash functions’, *Wikipedia*. Nov. 27, 2022. Accessed: Dec. 07, 2022. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Comparison_of_cryptographic_hash_functions&oldid=1124131327
[24] ‘SOURCE_DATE_EPOCH specification’. https://reproducible-builds.org/specs/source-date-epoch/ (accessed Dec. 07, 2022).
[25] ‘ReproducibleBuilds/StandardEnvironmentVariables - Debian Wiki’. https://wiki.debian.org/ReproducibleBuilds/StandardEnvironmentVariables#Checklist (accessed Nov. 07, 2022).
[26] ‘dpkg-parsechangelog(1) — dpkg-dev — Debian jessie — Debian Manpages’. https://manpages.debian.org/jessie/dpkg-dev/dpkg-parsechangelog.1.en.html (accessed Dec. 07, 2022).
[27] ‘Deterministic build systems — reproducible-builds.org’. https://reproducible-builds.org/docs/deterministic-build-systems/ (accessed Nov. 07, 2022).
[28] ‘Volatile inputs can disappear — reproducible-builds.org’. https://reproducible-builds.org/docs/volatile-inputs/ (accessed Nov. 07, 2022).
[29] ‘Value initialization — reproducible-builds.org’. https://reproducible-builds.org/docs/value-initialization/ (accessed Nov. 07, 2022).
[30] ‘Version information — reproducible-builds.org’. https://reproducible-builds.org/docs/version-information/ (accessed Dec. 07, 2022).
[31] ‘git-clone(1) - Linux man page’. https://linux.die.net/man/1/git-clone (accessed Dec. 07, 2022).
[32] ‘Git - git-config Documentation’. https://git-scm.com/docs/git-config (accessed Dec. 07, 2022).
[33] ‘Timestamps — reproducible-builds.org’. https://reproducible-builds.org/docs/timestamps/ (accessed Dec. 07, 2022).
[34] W. Hommel, ‘wolfcw/libfaketime’. Dec. 07, 2022. Accessed: Dec. 08, 2022. [Online]. Available: https://github.com/wolfcw/libfaketime
[35] ‘Make Mac bundles built with LXC match their KVM counterparts (#12240) · Issues · Legacy / Trac · GitLab’, *GitLab*. https://gitlab.torproject.org/legacy/trac/-/issues/12240 (accessed Dec. 07, 2022).
[36] ‘Timezones — reproducible-builds.org’. https://reproducible-builds.org/docs/timezones/ (accessed Dec. 07, 2022).
[37] ‘Locales — reproducible-builds.org’. https://reproducible-builds.org/docs/locales/ (accessed Dec. 07, 2022).
[38] ‘tar.git - GNU Tar’. https://git.savannah.gnu.org/cgit/tar.git/commit/?id=ef0f882382f6 (accessed Dec. 08, 2022).
[39] ‘perlsec - Perl security - Perldoc Browser’. https://perldoc.perl.org/perlsec#Algorithmic-Complexity-Attacks (accessed Dec. 08, 2022).
[40] ‘Debugging Options (Using the GNU Compiler Collection (GCC))’. https://gcc.gnu.org/onlinedocs/gcc/Debugging-Options.html#index-fdebug-prefix-map (accessed Dec. 08, 2022).
[41] ‘System images — reproducible-builds.org’. https://reproducible-builds.org/docs/system-

images/ (accessed Nov. 07, 2022).
[42] ‘Maven – Guide to Configuring for Reproducible Builds’. https://maven.apache.org/guides/mini/guide-reproducible-builds.html (accessed Nov. 07, 2022).
[43] ‘What’s in a build environment? — reproducible-builds.org’. https://reproducible-builds.org/docs/perimeter/ (accessed Nov. 07, 2022).
[44] ‘Recording the build environment — reproducible-builds.org’. https://reproducible-builds.org/docs/recording/ (accessed Dec. 08, 2022).
[45] ‘makepkg - ArchWiki’. https://wiki.archlinux.org/title/makepkg (accessed Dec. 08, 2022).
[46] ‘Definition strategies — reproducible-builds.org’. https://reproducible-builds.org/docs/definition-strategies/ (accessed Dec. 08, 2022).
[47] ‘MinGW-w64’. https://www.mingw-w64.org/ (accessed Dec. 08, 2022).
[48] ‘Build our own cctools for macOS cross-compilation (#9711) · Issues · Legacy / Trac · GitLab’, *GitLab*. https://gitlab.torproject.org/legacy/trac/-/issues/9711 (accessed Dec. 08, 2022).
[49] ‘ddmg.sh\build-helpers\gitian - builders/tor-browser-bundle - Old (2013-2017) build scripts for the Tor Browser Bundle based on gitian-builder’. https://gitweb.torproject.org/builders/tor-browser-bundle.git/tree/gitian/build-helpers/ddmg.sh (accessed Dec. 08, 2022).
[50] ‘Building from source — reproducible-builds.org’. https://reproducible-builds.org/docs/build-toolchain-from-source/ (accessed Dec. 08, 2022).
[51] ‘Virtual machine drivers — reproducible-builds.org’. https://reproducible-builds.org/docs/virtual-machine-drivers/ (accessed Dec. 08, 2022).
[52] ‘bitcoin/release-notes-0.9.4.md at 7fcf53f7b4524572d1d0c9a5fdc388e87eb02416 · bitcoin/bitcoin · GitHub’. https://github.com/bitcoin/bitcoin/blob/7fcf53f7b4524572d1d0c9a5fdc388e87eb02416/doc/release-notes/release-notes-0.9.4.md (accessed Dec. 08, 2022).
[53] ‘Reproducible Build Manager’. https://rbm.torproject.org/ (accessed Dec. 08, 2022).
[54] ‘runc’. Open Container Initiative, Dec. 08, 2022. Accessed: Dec. 08, 2022. [Online]. Available: https://github.com/opencontainers/runc
[55] ‘Docker overview’, *Docker Documentation*, Dec. 08, 2022. https://docs.docker.com/get-started/overview/ (accessed Dec. 08, 2022).
[56] ‘Provision development environments’, *Vagrant by HashiCorp*. https://www.vagrantup.com/use-cases/provision-development-environments (accessed Dec. 08, 2022).
[57] ‘diffoscope: in-depth comparison of files, archives, and directories’. https://diffoscope.org/ (accessed Dec. 08, 2022).
[58] ‘Rebuilderd - ArchWiki’. https://wiki.archlinux.org/title/Rebuilderd (accessed Dec. 08, 2022).
[59] ‘Arch Linux - archlinux-repro 20221114-1 (any)’. https://archlinux.org/packages/community/any/archlinux-repro/ (accessed Dec. 08, 2022).
[60] ‘Reproducible Builds / disorderfs · GitLab’, *GitLab*. https://salsa.debian.org/reproducible-builds/disorderfs (accessed Dec. 08, 2022).
[61] ‘Embedded signatures — reproducible-builds.org’. https://reproducible-builds.org/docs/embedded-signatures/ (accessed Nov. 07, 2022).
[62] ‘ReproducibleBuilds/BuildinfoSpecification - Debian Wiki’. https://wiki.debian.org/ReproducibleBuilds/BuildinfoSpecification#buildinfo_signatures (accessed Dec. 08, 2022).
[63] ‘ReproducibleBuilds - Debian Wiki’. https://wiki.debian.org/ReproducibleBuilds?action=recall&rev=83#Crude_bash_script_to_compare_two_binary_packages (accessed Nov. 07, 2022).