Short Paper: Estimating Patch Propagation Times across Blockchain Forks

Sébastien Andreina[1], Lorenzo Alluminio[2], Giorgia Azzurra Marson[1], and Ghassan Karame[3]

[1] NEC Labs Europe {sebastien.andreina,giorgia.marson}@neclab.eu
[2] Clearmatics, lorenzo.alluminio@clearmatics.com
[3] Ruhr-Universität Bochum, ghassan@karame.org

Abstract. The wide success of Bitcoin has led to a huge surge of alternative cryptocurrencies (altcoins). Most altcoins essentially fork Bitcoin's code with minor modifications, such as the number of coins to be minted, the block size, and the block generation time. In this paper, we take a closer look at Bitcoin forks from the perspective of vulnerability patching. By mining data retrieved from the GitHub repositories of various altcoin projects, we estimate the time it took to propagate relevant patches from Bitcoin to the altcoins. We find that, while the Bitcoin development community is quite active in fixing security flaws of Bitcoin's code base, forked cryptocurrencies are not as rigorous in patching the same vulnerabilities (inherited from Bitcoin). In some cases, we observe that even critical vulnerabilities, discovered and fixed within the Bitcoin community, have been addressed by the altcoins tens of months after disclosure.

1 Introduction

The wide success of Bitcoin has led to an explosion in the number of so-called "altcoins", i.e., cryptocurrencies designed as a fork of the Bitcoin-core code base. Altcoins exhibit minor differences to Bitcoin, e.g., some altcoins feature a different block-generation time (e.g., Dogecoin and Litecoin), other use a different hash function (e.g., Litecoin and Namecoin), or impose a different limit on the supply amount (e.g., Dogecoin, Litecoin). Despite these subtle differences, most altcoins share—to a large extent—the same technical foundations of Bitcoin. In the past decade, research has shown that Bitcoin (and many of its descendants) are vulnerable to a wide variety of attacks [15]. However, owing to strong development support, the Bitcoin-core software is routinely monitored and promptly patched (even including research results). Early studies about the security of proof-of-work blockchains [17] already hinted that some altcoins might offer weaker security compared to Bitcoin, owing to the ad-hoc parameters they adopt.

In this paper, we investigate the security of altcoins from the perspective of vulnerability patching. To this end, we select a set of important vulnerabilities reported in Bitcoin, and study how their patches were propagated through (Bitcoin-based) altcoins. Our approach relies on the inspection of GitHub repositories of popular cryptocurrencies, to identify relevant bugs and corresponding patches in the commit history of GitHub-hosted altcoin projects. Concretely, we study whether and how quickly various altcoin projects have addressed disclosed security issues. Unfortunately, retrieving detailed timing information associated to code changes in GitHub emerges as a challenging task. The reason is that most patches are taken directly from the main project repository and applied to the fork via a `rebase` operation which only

exposes a reliable timestamp for the original patch (applied to the main project), and not the actual time when the patch was ported (to the fork) [16]. Moreover, the original commits are no longer referenced after `rebase` occurs. As Git prunes unreferenced commits periodically, the timestamps associated to a given patch are lost with every subsequent `rebase` invocation. While prior studies on Bitcoin forks rely on code similarities to compare altcoins' software with Bitcoin Core [20, 21], they cannot infer patches that were ported via `rebase`.

To overcome this problem, we devised an automated tool to measure patch propagation times in Git-hosted forked projects even in the case of patches ported via `rebase`. Our tool leverages GitHub's event API and GH archive to estimate the time when a given patch is applied to a forked project. Namely, while GitHub follows the same practices as Git with pruning unreferenced commits, it keeps a log for all commits that ever existed. This information can be retrieved through GitHub's API, as long as one can reference the relevant commits. By traversing the graph of commits from the GH archive, we locate the (original) commit associated to the target patch and estimates the propagation time using three methods (cf. Section 2.2).

Leveraging our tool, we analyze the patch-propagation time of various altcoin projects, namely Dash [12], Digibyte [13], Monacoin [8], Litecoin [10], and Dogecoin [11], which we selected among GitHub-based open-source forks of Bitcoin to ensure diversity in terms of market cap, popularity, and vision. For each of the aforementioned altcoin projects, we estimate the time it took to apply relevant patches ported from Bitcoin. Specifically, we consider 47 patches comprising 11 vulnerabilities reported in academic papers, 23 Bitcoin's Common Vulnerabilities and Exposures (CVEs), 3 major CVEs in libraries used by Bitcoin, 3 Bitcoin improvement proposals and 7 major bugs found on the GitHub repository with tags related to the peer-to-peer network, covering crucial vulnerabilities reported in the last decade (see Table 1).

Our results (cf. Section 3) indicate that Bitcoin patched 55.3% of the vulnerabilities before their disclosure, while this number drops to 28.5%, 21.4%, 25%, 10.7% and 10.7%, for Litecoin, Dash, Dogecoin, Digibyte and Monacoin, respectively. For all selected altcoins, most patches have been applied with considerable delay compared to the disclosure time, thereby leaving many users vulnerable for several months or even years. We plan to release our tool as open-source to better aid the community in extracting timing information from re-based altcoins and other GitHub contributions[4]. Our results motivate the need to build automated analysis tools for forked Bitcoin projects in order to precisely explore whether a given vulnerability applies to any of those projects. This would indeed facilitate responsible disclosure of vulnerabilities to all affected forks prior to any publication of the vulnerability.

2 Measuring Patch Propagation Times in Git

Most altcoins port software patches (that have been applied to Bitcoin Core) via the `rebase` operations. Unfortunately, every `rebase` invocation modifies the history of the fork's repository—effectively altering the timestamps of all commits re-applied to the fork. In what follows, we study the problem of analyzing the propagation times of patches in Git across forked projects (i.e., the time to port a patch from the main project to the forked project), and introduce our tool, `GitWatch`, as an effective solution to this problem.

[4] `https://github.com/sebastien-an/GitWatch`

2.1 Git Operations

Commit. We define a commit as a pair $C = (M, D)$ of *metadata* and *data*—the latter indicates the applied changes. Metadata information is essential for examining the *history* of a repository. It includes a commit hash h (a.k.a. commit ID) that uniquely identifies the commit, a parent p referencing the previous commit, an author a and a committer c, and corresponding author timestamp t_a and committer timestamp t_c recording the creation time of the commit, respectively, the time when the commit was last modified. The commit ID h is a cryptographic hash over the changes D along with the remaining metadata: $h = H(p,a,c,t_a,t_c,D)$. Git allows associating *tags* to commit operations, e.g., to mark released versions of software. A tag τ contains a reference h to the target commit, a timestamp t, and a human-readable label.

Push. A "batch" of commits authored by a user u forms a sequence $(C_1,...,C_m)$ defined implicitly by the references to parent commits. Author and committer initially coincide with the user pushing the commits, and similarly author and committer timestamps coincide. Pushing a batch of commits $(C_1,...,C_m)$ triggers the addition of new snapshots, typically one per commit C_i, to the commit history $\mathcal{CH}$ of the repository.

Fork. A *fork* of an existing repository R is a repository R^χ that shares a common history with R—the latter is called the *main branch*. The latest commit that R and R^χ have in common is called *base commit*. Let $\mathcal{CH}$ and $\mathcal{CH}^\chi$ denote the commit histories of R and R^χ respectively. Then $\mathcal{CH} = (C_0,...,C_m,...,C_{m+s})$ and $\mathcal{CH}^\chi = (C_0,...,C_m,C_1^\chi,...,C_r^\chi)$, for $r > 0$, where C_m denotes the base commit and all commits C_i^χ diverge from the main branch.

Rebase. This operation allows integrating changes from the main branch R (e.g., Bitcoin) to a fork R^χ (e.g., an altcoin) by re-applying all commits pushed to R^χ starting from a new base commit in R—hence the term *rebase*. This operation is usually adopted to fetch the latest version of the original repository. Invoking `rebase` effectively "re-builds" the changes made in the fork on top of the new base commit, thereby modifying the commit history of the fork. Formally, let $\mathcal{CH} = (C_0,...,C_m,...,C_{m+s})$ and $\mathcal{CH}^\chi = (C_0,...,C_m,C_1^\chi,...,C_r^\chi)$ be the commit histories of the two repositories. Invoking `rebase` on R^χ with new base commit C_{m+k}, with $k > 0$, replaces $\mathcal{CH}^\chi$ with $(C_0,...,C_{m+k},C_1'^\chi...,C_r'^\chi)$, where each commit $C_i'^\chi$ updates the original commit C_i^χ adapting the metadata to the new base commit C_{m+k}—the committed changes D remain the same. This update modifies the first parent commit which, in turn, triggers a chain reaction and modifies all subsequent parent commits. Formally:

$$C_1'^\chi.p \leftarrow C_{m+k}.h \quad \wedge \quad C_i'^\chi.p \leftarrow C_{i-1}'^\chi.h \quad \forall i = 2,...,r. \tag{1}$$

Rebasing has the crucial effect of updating the committer timestamp with the current time (while author timestamps remain unchanged):

$$C_i'^\chi.t_c \leftarrow \text{current time} \quad \wedge \quad C_i'^\chi.t_a = C_i^\chi.t_a. \tag{2}$$

Rebasing makes timestamps unreliable. Rebasing can cause the loss of relevant timing information in the case of multiple `rebase` operations being performed on the same repository. Every `rebase` invocation preserves the author timestamp t_a of the original commits, however, it resets all committer timestamps t_c in the commit history to the current time—thereby overwriting all timestamps of previous `rebase` operations. This behavior is illustrated in Figure 1. After a rebase, the old commits $C_1^\chi,...,C_r^\chi$ become unreferenced and are "dangling". For saving up space, dangling commits are automatically pruned by Git. However, when a rebase replaces C_i with C_i', the two commits are factually different

(due to differing metadata) and are initially both accessible in GitHub via their respective commit IDs. Assuming no pruning, this observation provides us with a strategy to retrieve the timestamp of rebases: by listing all the different versions of a commit C_i and their respective committer timestamp. Our methodology (c.f. Section 3) is based on this intuition to estimate the timing of rebases, yet it is compatible with the pruning of dangling commits.

2.2 Extracting rebase timing

GitHub generates events for all operations on subscribable public repositories. To extract meaningful information about patch propagation time—even when the patch is applied via rebasing—we rely on two main resources: GitHub's event API and GH archive. GH archive [14] is an openly accessible service that provides the history of all GitHub events since 2011. We observe that while rebases create dangling commits that are not retrieved when cloning, these commits can still be queried through GitHub's API by requesting the corresponding hash. Our tool, `GitWatch`, relies on this to retrieve the timestamps of dangling commits.

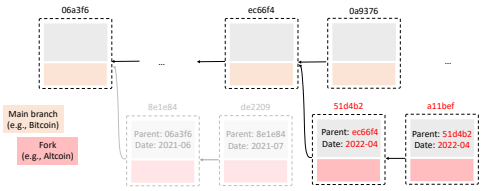


Fig. 1: Effect of `rebase` on commit metadata: commit ID, parent commit, and committer date are modified. Dotted boxes represent commits, arrows point to parent commits.

To measure the patch propagation times for a GitHub project χ, `GitWatch` first reconstructs the tree of commits that ever existed in χ, building a graph $\mathcal{G}_\chi = (V,E)$ containing all commits C in χ (including dangling commits) as vertices, and with edges representing the parent to child relationship, i.e., $C \in V$ and $(C.p,C) \in E$. To do so, it crawls GH archive for all events pertaining to χ in order to retrieve all commits from GitHub's API. Using $\mathcal{G}_\chi$, `GitWatch` locates the commit (if any) applying a target patch to χ and estimates the corresponding timestamp using three different heuristics: patch-commit finder (PCF), patch-event finder (PEF), and patch-tag finder (PTF). The patch propagation time Δ, from Bitcoin to the altcoin, is then derived by comparing these estimates with the original timestamp of the Bitcoin patch. The reliance on all three heuristics helps in eliminating possible false positives that may arise due to missing events in the GH archive. Whenever we obtain different results from the heuristics, `GitWatch` returns the smallest timeframe by default.

Patch-commit finder (PCF). Here, given a patch commit $C_i \in \mathcal{CH}^{BC}$, we traverse $\mathcal{G}_\chi$ to collect all non-Bitcoin commits C_j containing C_i in their history. Concretely, we construct a list $\text{nbcc}_\chi(C_i)$ of "non-Bitcoin child commits" defined as follows:

$$\begin{aligned} C_j \in \text{nbcc}_\chi(C_i) \iff & C_j \notin \mathcal{CH}^{BC} \wedge C_j \in \mathcal{CH}^{\chi} \wedge \exists (E_1,..,E_n) \in \mathcal{G}_\chi : \\ & E_1.\texttt{from} = C_i \wedge E_n.\texttt{to} = C_j \wedge \forall i \in [1,n-1],\, E_i.\texttt{to} = E_{i+1}.\texttt{from}. \end{aligned} \tag{3}$$

PCF then locates in $\text{nbcc}(C_i)$ the earliest commit C^* such that:[5]

$$C^* \in \text{nbcc}_\chi(C_i) \wedge C^*.t_c \leq C_j.t_c \; \forall C_j \in \text{nbcc}_\chi(C_i). \tag{4}$$

The estimated patch-propagation time is $\Delta^{\text{PCF}} \leftarrow C^*.t_c - C_i.t_a$.

Patch-event finder (PEF). In addition to $\mathcal{G}_\chi$, our second heuristic relies on inspecting GitHub events. Events are associated to one or more commits: a `push` event e contains the list of commits $(C_1,...,C_m)$ pushed by the author, i.e., $e = (\boldsymbol{C_e}, t_e)$ with $\boldsymbol{C_e} = (C_1,...,C_m)$ and t_e is

[5] C^* following this property may not be unique, as multiple commits can have the same timestamp.

the event timestamp. We denote by ϕ the mapping from commits to events, i.e., $\phi(C_i) := e$ for all $i = 1,...,m$. Slightly abusing notation, we write $C_i \in e \Leftrightarrow \phi(C_i) = e$. Similarly to PCF, we look for the earliest non-Bitcoin commit C^* that contains patch C_i in its history, however, we measure elapsed time with respect to event timestamps rather than commit timestamps. We estimate the patch propagation time as the time span between the creation of the patch commit C_i and the oldest event that references a commit C_j that has C_i in its history. Let $\mathcal{E}_\chi := \{e \mid \exists C \in \mathcal{CH}^\chi : \phi(C) = e\}$ denote the set of events pertaining to the altcoin χ and recorded in the GH archive. PEF characterizes a relevant commit C^* as follows:

$$C^* \in \text{nbcc}_\chi(C_i) \wedge \phi(C^*) \in \mathcal{E}_\chi \wedge \phi(C^*).t \leq \phi(C_j).t \; \forall C_j \in \text{nbcc}_\chi(C_i) \cap \mathcal{E}_\chi. \tag{5}$$

The estimated patch propagation time is $\Delta^{\text{PEF}} \leftarrow \phi(C^*).t - C_i.t_a$.

Patch-tag finder (PTF). Our third heuristic relies on timestamps recorded for relevant tags. Intuitively, we estimate the patch propagation time as the interval between the creation of the Bitcoin patch C_i and the creation of the first non-Bitcoin tag associated to a commit in χ that has C_i in its history. We define "non-Bitcoin child tag" analogously to that of non-Bitcoin child commit:

$$\tau \in \text{nbct}_\chi(C_i) \Longleftrightarrow \tau.h \in \text{nbcc}_\chi(C_i) \wedge \tau \in Tags_\chi \wedge \tau \notin Tags_{BC}. \tag{6}$$

PTF identifies a relevant tag $\tau^* \in \text{nbct}_\chi(C_i) \wedge \tau^*.t \leq \tau.t \; \forall \tau \in \text{nbct}_\chi(C_i)$, and estimates the patch propagation delay as $\Delta^{\text{PTF}} \leftarrow \tau^*.t - C_i.t_a$.

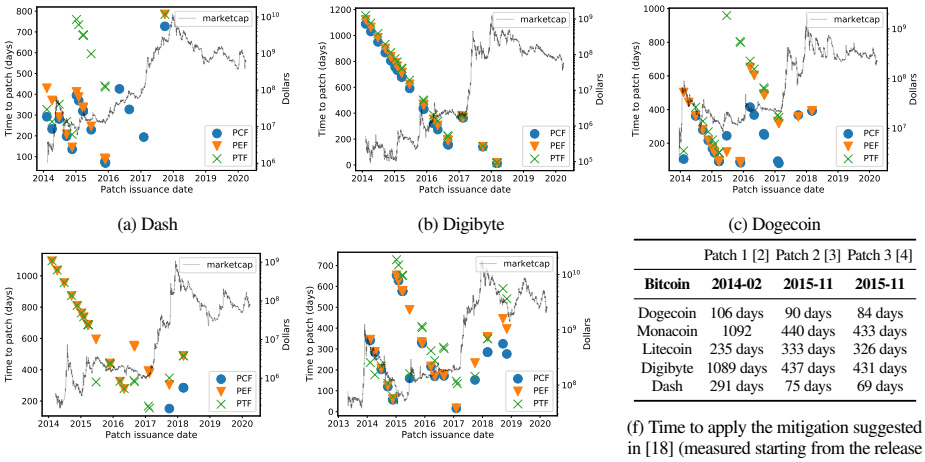
Comparison. Assuming successful retrieval of all dangling commits, our methodology lists all different versions of the relevant commit C^* (one per rebase), allowing us to select the most accurate commit timestamp. Our three heuristics therefore overcome the problem of retrieving reliable timestamps in the presence of rebasing (c.f. Section 2.1). The major limitation of PCF and PEF is that they may under-approximate the patching time in case a developer creates the patch locally (or on a dev branch) and pushes the patch to the main branch at a later time (e.g., for testing the patched code locally). PCF further relies on the developer's local clock, which could be skewed. PTF is not affected by these limitations, as it outputs the most conservative timestamp. We therefore expect PTF to output the most accurate estimate in typical scenarios, in particular because most users do not compile the latest modifications based on the current version of the main branch (which may be unstable); they are more likely to use released versions of the code, which are marked with tags. Notice that PCF, PEF, and PTF exclusively inspect commits that are either rebased or merged with the same code base: patches introduced with a different code base may therefore not be identified.

3 Methodology & Evaluation

3.1 Dataset

In our evaluation, we restrict our analysis to bugs and patches related to Bitcoin, in particular, how they are propagated through altcoins that are based on the same code base. Since we are interested in patches that are not specific to Bitcoin but relevant to most altcoins, we mainly focus on reported bugs on the peer-to-peer layer as this layer is generally inherited by altcoins (including those that introduce non-negligible modifications to the code base).

We analyze five altcoins, which we selected among existing open-source forks of Bitcoin: Dash [12], initially known for its early adoption in darknet markets, currently worth 2.65 Billion USD; Digibyte [13], a cryptocurrency advertised for its improved functionality and security, currently worth 1.12 Billion USD; Monacoin [8], aimed to become a national



(a) Dash (b) Digibyte (c) Dogecoin

(d) Monacoin (e) Litecoin

	Patch 1 [2]	Patch 2 [3]	Patch 3 [4]
Bitcoin	**2014-02**	**2015-11**	**2015-11**
Dogecoin	106 days	90 days	84 days
Monacoin	1092	440 days	433 days
Litecoin	235 days	333 days	326 days
Digibyte	1089 days	437 days	431 days
Dash	291 days	75 days	69 days

(f) Time to apply the mitigation suggested in [18] (measured starting from the release of the Bitcoin patch).

Fig. 2: Time for a patch issued by Bitcoin-core to be included to the different altcoins. In the plots, the blue circle, the orange triangle, and the green cross represent respectively the output values given by the graph, the event, and the tag approach; the market capitalization over time of each coin is plotted as a black dotted line against the values on the right y-axis.

payment system in Japan, currently worth 0.11 Billion USD; Litecoin [10] and Dogecoin [11], which emerges among the most popular first-generation derivatives of Bitcoin with a market capitalization of 14.82 Billion USD and 40.07 Billion USD respectively. We then selected a list of 47 Bitcoin commits, 11 representing patches suggested by top-tier publications [18, 19, 22], 23 representing patches of CVEs, 3 representing Bitcoin improvement proposals (BIP), 3 representing CVEs in libraries used by Bitcoin and the remaining 7 representing bugs found on the GitHub repository with tags related to the peer-to-peer network. These patches include the majority of network and peer-to-peer vulnerabilities that were reported in the last decade.

3.2 Validation of `GitWatch`

To validate the effectiveness of `GitWatch`, we manually identified publication dates of relevant patches (by investigating release notes), and we compared these dates with the output of `GitWatch` for the same vulnerability. Our results, shown in Figure 2f, confirm that for all the ground-truth data points we found, the actual patching time falls within the interval reported by the three heuristic used by `GitWatch` (i.e., between the minimum and maximum estimated propagation time). As expected, PTF provides the most accurate results, especially since release notes are usually part of a new release to which a tag is assigned.

Table 2: Ground-truth data to validate `GitWatch`.

Vulnerability	Altcoin	Time	PCF	PEF	PTF
BIP 65	Litecoin	179 days [6]	159	160	181
BIP 65	Dogecoin	958 days [7]	244	147	958
BIP 66	Dogecoin	194 days [5]	142	142	194
CVE-2013-4627	Litecoin	33 days [1]	17	45	18
CVE-2013-4165	Litecoin	28 day [1]	10	529	13

3.3 Evaluation results

As shown in Figure 2, `GitWatch` provides consistent timing estimates, which converge in most cases. Dash (Figure 2a) appears to port patches more quickly, compared to the other blockchains, most of the times with a delay between 200 and 400 days. Dogecoin and Litecoin instead (Figures 2c and 2e) show more variable patching delays,

Table 1: Estimated patching time (in days) based on our dataset. A dash (-) indicates that `GitWatch` could not find the patch in the altcoin.

Name	Pub Date	Description	Bitcoin	Litecoin	Dash	Dogecoin	Digibyte	Monacoin
		Total Number of fixes	47/47	41/42	21/28	23/28	25/28	26/28
Paper [19]	2015-08-14	deterministic random eviction	-143	19	15	92	681	684
Paper [19]	2015-08-14	random selection sha1	-143	19	15	92	681	684
Paper [19]	2015-08-14	random selection sha2	-143	19	15	92	681	684
Paper [19]	2015-08-14	test before evict	935	285	-	392	13	285
Paper [19]	2015-08-14	feeler connections	375	58	327	256	162	165
Paper [19]	2015-08-14	more buckets	-143	19	15	92	681	684
Paper [19]	2015-08-14	more outgoing connections	1482	-	-	-	-	-
Paper [18]	2015-10-16	no inv messages	44	326	69	84	430	433
Paper [18]	2015-10-16	filtering inv by ip address	38	333	75	90	437	440
Paper [18]	2015-10-16	penalizing non-responding nodes	-615	235	291	106	1089	1092
Paper [22]	2012-10-18	forward double spending attempts	617	202	281	362	950	953
Vulnerability	-	limit the number of IP learned from each DNS	0	103	-	392	13	103
Vulnerability	-	ensure tried table collisions eventually get resolved	0	281	-	-	-	-
GitHub bug	-	fixes fee estimate and peers files only when initialized	0	119	198	279	867	870
GitHub bug	-	check block header when accepting headers	0	56	135	216	804	808
GitHub bug	-	introduce block download timeout	0	8	87	168	756	759
GitHub bug	-	de-serialization bug where AddrMan is corrupted	0	169	426	367	273	276
GitHub bug	-	don't deserialize nVersion into CNode	0	15	194	94	376	167
CVE-2014-0160	2014-04-07	Remote memory leak via payment protocol	1	176	233	0	1030	1034
BIP 66	2015-02-13	FakeConf: Strict DER signatures	-12	4	61	142	731	734
BIP 65	2015-11-12	FakeConf: OP_CHECKLOCKTIMEVERIFY	-143	159	229	147	591	322
CVE-2016-10724	2018-07-02	DoS: Alert memory exhaustion	-836	216	-	414	320	323
CVE-2018-17144	2018-09-20	Inflation: Missing check for duplicate inputs	-3	1	1	-	155	1
CVE-2017-18350	2019-06-22	Buffer overflow from SOCKS proxy	-632	151	727	91	140	151
CVE-2018-20586	2019-06-22	Deception: Debug log injection	-229	41	380	-	106	244
CVE-2014-0224	2014-06-05	OpenSSL CVE	0	118	174	0	972	975
CVE-2018-12356	2018-06-14	Regex bug	1	184	-	291	50	184
CVE-2019-6250	2019-01-13	Vulnerability in the ZeroMQ libzmq library	5	31	-	-	-	31
		Average	7.53	114.85	188.0	185.17	519.55	503.3

ranging between 50-600 days, respectively, and 100-500 days on average. Digibyte and Monacoin (Figures 2b and 2d) exhibit an apparent linearly decreasing delay. This peculiar behavior suggests that `rebase` operations to import the Bitcoin's patches are executed on a regular pace, in a manner that appears to be decoupled from the actual patch release. This would explain the downward trend in the plots, indicating that groups of patches are actually ported on the corresponding fork at the same time. To summarize, all five analyzed altcoins apply patches with a delay between several months to a few years. We include the detailed results of our study in Table 1. Out of the 47 selected commits, we omitted 5 CVEs that were patched before any of the altcoins were created, and 13 CVEs and 1 BIP where only Litecoin was released. Those 14 patches were ported by Litecoin with an average delay of 97 days. Our results show that Bitcoin issues patches to most critical vulnerabilities and CVEs in a prompt manner, often before the publication of vulnerability (i.e., in compliance with the responsible disclosure process).

4 Case Studies

We now look more closely at two specific vulnerabilities found in Bitcoin, which we selected because they are prominent and recent (disclosure in 2015 and 2017 resp.).

Case Study 1: Tampering with the Delivery of Information in Bitcoin [18]. In order to sustain higher throughput and scalability, Bitcoin implemented a number of optimizations and scalability measures. In [18], it was shown that some of those measures come at odds with the security of the system. As a direct outcome of this vulnerability, a resource-constrained adversary could mount a large-scale Denial-of-Service attack on Bitcoin—effectively halting

the delivery of all blocks and transactions in the system. The authors suggested various improvements that resulted in multiple patches:

- Patch 1 - f59d… [2], penalizing nodes that do not respond to block requests.
- Patch 2 - 5029… [3], preventing adversaries from filling up the advertisement table.
- Patch 3 - 5026… [4], replacing the advertisement message with the full block header.

As shown in Figure 2f, Dash and Dogecoin took almost 3 months to port these patches from Bitcoin, while Monacoin, Litecoin and Digibyte required between 7 months and 3 years.

Case Study 2: CVE-2017-18350 [9]. This buffer-overflow vulnerability of the Bitcoin-core software was located in the proxy support, and would enable a malicious proxy server to overwrite the program stack, allowing it to perform remote code execution. However, to be vulnerable, the wallet needs to be configured to use a malicious proxy, therefore reducing the general risk on the users. Since remote code execution could allow any third party full access to the machine running the node, we deemed that this CVE to be of particular interest due to its potential drastic impact. This vulnerability was discovered on September 21st 2017, was patched two days later, on September 23rd and the patch was merged with the main branch of the Bitcoin-core repository four days later on September 27th, 2017. To give enough time to the users for applying the patch, the CVE itself was published only on the June 22nd, 2019. While this patch was applied directly to most of the different altcoins based on the Bitcoin-core software, Dash [12] only patched it several months after it was published, on November 19th, 2019. Dash users were seemingly using a vulnerable software with no available update for several months after the disclosure of the vulnerability. Dogecoin, Digibyte, Monacoin and Litecoin took respectively 91 days, 140 days, 151 days and 151 days to patch this vulnerability after it was discovered. While they all patched it before the vulnerability was disclosed, the software still remained unpatched for several months.

5 Conclusion

In this paper, we showed that various altcoins exhibit weaker stability compared to Bitcoin Core. Beyond confirming the folklore result that patch propagation is slow for some altcoins, we introduced a new technique to estimate the time for altcoins to propagate security patches, and deterimne which altcoins are faster in adopting a patch. For instance, Dash patched CVE-2017-18350 5 months after the public release of the CVE. Moreover, among the five altcoins we analyzed (some of which are worth several billions), Litecoin is the only project to have consistently ported patches within 1 year of their release. Our findings also suggest that a good number of network vulnerabilities pointed out in academic research were patched in the core Bitcoin protocol and also eventually in many altcoins. We hope that our work further motivates the need for a proper responsible disclosure of vulnerabilities to all forked chains prior to any publication of the vulnerability.

Acknowledgments

This work was partially funded by the Deutsche Forschungs-gemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972 and the European Union (INCODE, Grant Agreement No 101093069). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Litecoin v0.8.4.1 release note. https://litecoinmirror.wordpress.com/2013/09/04/litecoin-0-8-4-1-release-notes/amp/ (2013)
2. Bitcoin patch propagation 1. https://github.com/bitcoin/bitcoin/commit/f59d8f0b644d49324cabd19c58cf2262d49e1392 (2014)
3. Bitcoin patch propagation 2. https://github.com/bitcoin/bitcoin/commit/5029698186445bf3cd69d0e720f019c472661bff (2015)
4. Bitcoin patch propagation 3. https://github.com/bitcoin/bitcoin/commit/50262d89531692473ff557c1061aee22aa4cca1c (2015)
5. Dogecoin v1.10 release note. https://github.com/dogecoin/dogecoin/releases?q=BIP66&expanded=true (2015)
6. Litecoin v0.10.4 release note. https://github.com/litecoin-project/litecoin/blob/v0.10.4.0/doc/release-notes-litecoin.md (2015)
7. Dogecoin v1.14 alpha release note. https://github.com/dogecoin/dogecoin/releases/tag/v1.14-alpha-1 (2018)
8. Monacoin. https://monacoin.org/ (2018)
9. Cve-2017-18350. https://medium.com/@lukedashjr/cve-2017-18350-disclosure-fe6d695f45d5 (2019)
10. Litecoin. https://litecoin.com/en/ (2020)
11. Dogecoin. https://dogecoin.com/ (2021)
12. Dash. https://www.dash.org/ (2022)
13. Digibyte. https://digibyte.org/en-us/ (2022)
14. Gh archive. https://www.gharchive.org/ (2022)
15. Böhme, R., Eckey, L., Moore, T., Narula, N., Ruffing, T., Zohar, A.: Responsible vulnerability disclosure in cryptocurrencies. Commun. ACM **63**(10), 62–71 (sep 2020). https://doi.org/10.1145/3372115, https://doi.org/10.1145/3372115
16. Businge, J., Moses, O., Nadi, S., Berger, T.: Reuse and maintenance practices among divergent forks in three software ecosystems. In: Empirical Software Engineering (2021)
17. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 3–16. ACM (2016). https://doi.org/10.1145/2976749.2978341, https://doi.org/10.1145/2976749.2978341
18. Gervais, A., Ritzdorf, H., Karame, G.O., Capkun, S.: Tampering with the delivery of blocks and transactions in bitcoin. In: Ray, I., Li, N., Kruegel, C. (eds.) Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015. pp. 692–705. ACM (2015). https://doi.org/10.1145/2810103.2813655, https://doi.org/10.1145/2810103.2813655
19. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: Jung, J., Holz, T. (eds.) 24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015. pp. 129–144. USENIX Association (2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman
20. Hum, Q., Tan, W.J., Tey, S.Y., Lenus, L., Homoliak, I., Lin, Y., Sun, J.: Coinwatch: A clone-based approach for detecting vulnerabilities in cryptocurrencies. In: IEEE International Conference on Blockchain, Blockchain 2020, Rhodes, Greece, November 2-6, 2020. pp. 17–25. IEEE (2020). https://doi.org/10.1109/Blockchain50366.2020.00011, https://doi.org/10.1109/Blockchain50366.2020.00011

21. Jia, A., Fan, M., Xu, X., Cui, D., Wei, W., Yang, Z., Ye, K., Liu, T.: From innovations to prospects: What is hidden behind cryptocurrencies? In: MSR. pp. 288–299. ACM (2020)
22. Karame, G., Androulaki, E., Capkun, S.: Double-spending fast payments in bitcoin. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012. pp. 906–917. ACM (2012). https://doi.org/10.1145/2382196.2382292, https://doi.org/10.1145/2382196.2382292