Certificate Transparency Revisited: The Public Inspections on Third-party Monitors

Aozhuo Sun*‡ Jingqiang Lin†✉ Wei Wang* Zeyan Liu§ Bingyu Li¶ Shushang Wen† Qiongxiao Wang‖ Fengjun Li§
*Institute of Information Engineering, Chinese Academy of Sciences, China
†School of Cyber Science and Technology, University of Science and Technology of China, China
‡School of Cyber Security, University of Chinese Academy of Sciences, China
§The University of Kansas, USA
¶School of Cyber Science and Technology, Beihang University, China
‖Beijing Certificate Authority Co., Ltd, China
Email: sunaozhuo@iie.ac.cn, linjq@ustc.edu.cn, wangwei@iie.ac.cn, zyliu@ku.edu,
libingyu@buaa.edu.cn, sswen@mail.ustc.edu.cn, wangqiongxiao@bjca.org.cn, fli@ku.edu

***Abstract*—The certificate transparency (CT) framework has been deployed to improve the accountability of the TLS certificate ecosystem. However, the current implementation of CT does not enforce or guarantee the correct behavior of third-party monitors, which are essential components of the CT framework, and raises security and reliability concerns. For example, recent studies [32], [33] reported that 5 popular third-party CT monitors cannot always return the complete set of certificates inquired by users, which fundamentally impairs the protection that CT aims to offer. This work revisits the CT design and proposes an additional component of the CT framework, *CT watchers*. A watcher acts as an *inspector of third-party CT monitors* to detect any misbehavior by inspecting the certificate search services of a third-party monitor and detecting any inconsistent results returned by multiple monitors. It also semi-automatically analyzes potential causes of the inconsistency, e.g., a monitor's misconfiguration, implementation flaws, etc. We implemented a prototype of the CT watcher and conducted a 52-day trial operation and several confirmation experiments involving 8.26M unique certificates of about 6,000 domains. From the results returned by 6 active third-party monitors in the wild, the prototype detected 14 potential design or implementation issues of these monitors, demonstrating its effectiveness in public inspections on third-party monitors and the potential to improve the overall reliability of CT.**

I. Introduction

The TLS certificate ecosystem has been built on the trust of the certificates issued by trusted certification authorities (CAs) to domain owners. Browsers verify these certificates to establish secure TLS sessions. Unfortunately, numerous incidents involving the generation and use of bogus certificates have been consistently reported [48], [6], [16], [22], [35], [59], showing that incorrect or bogus certificates were either misissued by CAs due to operational errors [28] or generated by malicious parties through various attacks [4], [11], [12].

To tackle this problem, the certificate transparency (CT) framework has been developed. It consists of *log servers*, *auditors*, and *monitors*, as shown in Fig. 1. Its primary goal is to promptly detect misissued certificates [5], [29], thereby enhancing the accountability of certificate signing services. The CT framework does not rely on any single CA, log server, or auditor since these components could be flawed or potentially malicious. Instead, it records each issuance action of the CAs in publicly visible logs at the log servers. For each recorded certificate, the log server generates a signed certificate timestamp (SCT) as a form of assurance, indicating that the certificate will be publicly logged. Meanwhile, the recording operations of the log servers are audited by distributed auditors, who work collaboratively to ensure log servers deliver the promised logging services. With a modest number of independent log servers and auditors, CT builds a distributed append-only ledger to record all TLS certificates that CT-compliant browsers accept.

However, a CT-compliant browser will accept any fraudulent certificate that has been logged in the CT log and acquired a valid SCT. To detect bogus certificates, CT monitors are employed, which retrieve certificates from the logs and identify all certificates for a given domain, e.g., through a certificate search service. If a CT monitor fails to return bogus certificates, whether due to negligence or intentional actions, the effectiveness of the CT framework is compromised. Unfortunately, recent studies [32], [33] showed that CT monitors may not always return *all* the certificates for an inquired domain, raising concerns that misissued and fraudulent certificates can evade monitors' detection due to potential design and implementation defects in their certificate search services. This calls for effective inspection and detection solutions for third-party monitors to ensure that they provide reliable and timely services. However, to our best knowledge, there is no mechanism in CT to inspect monitors' operations [24], [29].

We revisit the CT framework design and propose a new component, called *watchers*, to inspect third-party monitor services. Similar to CT auditors that are tasked with detecting misbehavior on the logs, watchers are expected to detect misbehavior on third-party monitors, including faulty services and malicious actions. The watchers enable public inspections on third-party monitors and their certificate search services,

Jingqiang Lin is the corresponding author.
Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.24834
www.ndss-symposium.org

which is essential for detecting bogus certificates. Therefore, designing watchers to improve the security promise that CT offers to the TLS certificate ecosystem is highly non-trivial.

CT watchers can be operated by any interested party in the TLS ecosystem such as domain owners, CAs, monitors, etc. Based on their needs and capacities, they can run either a *light watcher* that supports automated inspections of monitors' search services for a small set of domains, or a *full watcher* that monitors a large set of domains with an additional semi-automated fault analysis function. In particular, for each domain in the target set, both watchers will (*i*) retrieve certificates from multiple monitors to construct a reference set as an approximation of the ground truth and (*ii*) identify any inconsistency between the result returned by each monitor and the reference set to detect faulty or malicious monitors. The semi-automated fault analysis of the full watchers will further identify trigger features in certificates that potentially cause a monitor to mishandle these certificates. The trigger features and error locations within the operational process of these monitors are useful for identifying their internal bugs.

According to the threat model of CT, the watchers should not be fully trusted. They can be malicious, but cannot cause any honest party in CT to deviate from the protocol. An honest watcher always constructs the correct reference set for a given domain and verifies the trigger features shared by other watchers. Therefore, they can work cooperatively to inspect the services provided by third-party monitors without introducing additional trusted third parties.

We implemented and open-sourced prototypes for both light and full watchers, which inspect 6 most popular and currently active third-party monitors, i.e., Censys, crt.sh, Entrust Search, Facebook Monitor, Google Monitor, and SSLMate Spotter. We refer the readers to Appendix F for more details. We performed a trial operation (January 25 - March 16, 2020) for 4,000 randomly selected domains and uncovered several implementation bugs and design limitations in six monitors. The results showed that the light watcher can promptly inspect third-party monitors with very limited resources, while the full watcher can effectively detect defects in the monitors' search service. We discussed standardization suggestions for the services of third-party monitors, which can significantly reduce the watchers' operational overhead. Our contributions are three-fold: (*a*) We proposed CT watchers for inspecting the services of third-party monitors, which improves the overall security of CT. (*b*) We designed and implemented automated light watchers and semi-automated full watchers. Our evaluation of the operating cost of the light watcher shows its practicality. Finally, (*c*) we conducted real-world experiments including a 52-day trial operation, and validated the effectiveness of CT watchers. We discovered several design and implementation flaws/limitations in 6 inspected monitors and disclosed them to the vendors.

The remainder of the paper is organized as follows. Section II presents the reliability problem of third-party monitors and Section III analyzes the related security threats. Sections IV and V present the detailed design and implementation of the watchers, respectively. Section VI reports the trial operation and the discovered flaws. We discuss the standardization suggestions and future work in Section VII, present the related works in Section VIII, and conclude the work in Section IX.

II. Uninspected Third-party Monitors of CT

A. Certificate Transparency

The main goal of CT [5], [29] is to increase the visibility of certificates by requiring the CAs to place the issued certificates in public logs (in the *logging* phase) so that the users such as domain owners could actively check the certificates with the assistance of CT monitors to discover the bogus ones (in the *checking* phase). As shown in Figure 1, CT introduces three new components into the traditional public key infrastructure (PKI) system:

Log server. When issuing a certificate, the CA is required to submit it to several log servers [19], [2]. The log server records the certificate in its publicly visible log within the maximum merge delay (MMD) and returns an SCT as a promise and verifiable proof. All received certificates are organized into a Merkle hash tree to ensure that they are append-only. In addition, the root node is periodically signed by the log server, called the signature tree head (STH).

Auditor. Auditors are lightweight software components that audit log servers to ensure they consistently fulfill their promises[1]. They verify whether logs are append-only by comparing two STHs and confirm that each SCT corresponds to a record in the logs by validating the audit path, which is the sequence of the siblings of all nodes on the path from the node of the record to the root node.

Monitor. CT participants retrieve certificates of interest from the log server via the monitor, see subsection II-B for details. Monitors can be a domain owner that searches certificates by itself (i.e., *self-monitor*) or a service providing certificate search and notification services (i.e., *third-party monitor*).

In TLS negotiations, a *CT-compliant* browser only accepts certificates with SCTs. There are two methods for delivering an SCT with a certificate. (*a*) *X.509 v3 extension:* Before a CA issues a certificate, it creates a *precertificate*, which binds the same data but is formatted differently from the final certificate. Then, the CA submits the precertificate to log servers and gets SCTs. Finally, the certificate is issued with the SCTs embedded as a certificate extension. (*b*) *TLS extension or OCSP Stapling:* After a CA issues a certificate for a website, the CA or the website submits it to log servers and gets SCTs. The website uses a TLS extension or OCSP extension to deliver the SCT to the client during the TLS handshake.

B. Third-party Monitors in the Wild

Operating a monitor requires non-trivial resources. For example, it needs to handle newly-appended log entries every day, parse non-conforming certificates [47], and maintain the list of logs to monitor [50]. This involves handling about 6 million and 15 million new records (including precertificates and certificates) daily in 2018 and 2022, respectively [32], [39]. Thus, in practice, many users rely on professional third-party monitors to search for certificates of interest. To the best of our knowledge, there are 12 third-party monitors on the Internet,

[1]Some browser vendors such as Chrome [52], [38] implement a standalone auditor to verify audit paths on behalf of all browsers.

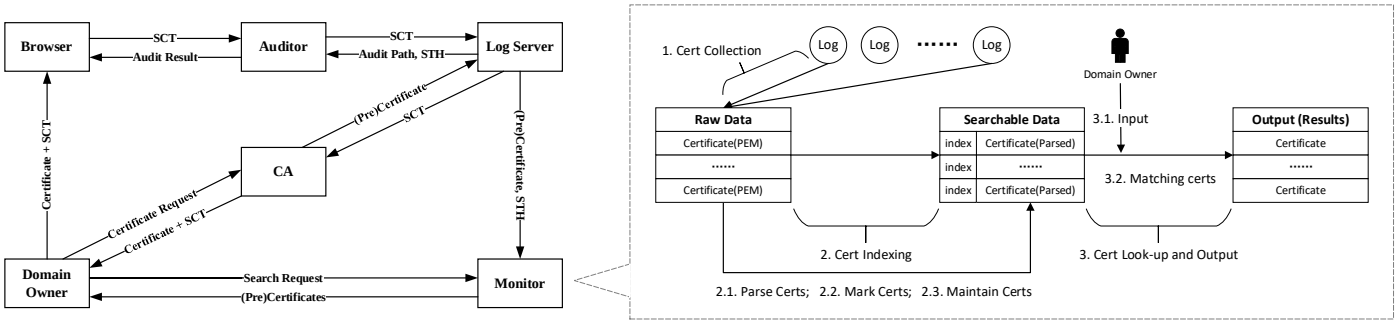


Fig. 1: The framework of CT and the workflow of the third-party monitor.

as listed in TABLE I. Among them, 6 monitors provide active certificate search services[2], which are studied in this work.

A third-party monitor fetches certificates in the public logs and indexes them to provide search and/or subscription services to users. As shown in Fig. 1, it pre-selects a set of logs to be monitored and operates three basic functions [49], [9], [29], known as *certificate collection*, *indexing*, and *look-up*. To collect data, it periodically fetches newly-appended certificate entries from each log, and then parses the raw certificate records to extract the domain name(s), i.e., common name (CN) and dNSNames in the subject alternative name extension (SAN:dNSNames), as keywords[3] to index each certificate. Other information about a certificate (e.g., the SHA256 fingerprint and the time submitted to logs) is also parsed and saved in the certificate indexing. Finally, when receiving an inquired domain, it looks up the certificates with matching keywords and returns the result.

C. Unreliable Certificate Search Services

We define the *complete set* of the certificates for an inquired domain [32] (e.g., z.y.x) as all the *unexpired* certificates issued for (*a*) the domain z.y.x; (*b*) all its subdomains with or without wildcard (e.g., www.z.y.x, *.z.y.x and *.game.z.y.x); and (*c*) its parent domain with a wildcard (e.g., *.y.x) if the parent domain is *not* an effective top-level domain (eTLD) [36]. All these certificates are called *relevant certificates* in this paper. Moreover, the comparison is conducted with SAN:dNSName in a case-insensitive manner[4].

A reliable monitor should return the complete set of the relevant certificates and in a *timely* manner, which means the maximum delay between a certificate's appearance in the public log and in the search result returned by the monitor should not be large. Otherwise, the delay could be exploited by potential man-in-the-middle or impersonation attacks using misissued bogus certificates.

However, recent studies [32], [33] experimentally showed that several well-known monitors did not achieve the expected reliability by timely returning the complete results. In their experiments with 6,000 randomly selected domains, none of the evaluated monitors returned the complete results for all the inquired domains and 12.6% - 52.3% of the relevant certificates were missing in the returned results. Besides, some monitors returned *irrelevant* certificates [32], [33], e.g., Censys returned code-signing certificates whose CN is not a valid domain name but contains the inquired domain.

III. Security Threats from Third-Party Monitors

The current CT design (e.g., SCTs, auditors, the gossip protocol [37]) focuses mainly on the logging phase to ensure its security against malicious logs that might even collude with a malicious CA. Unfortunately, the security of the checking phase is overlooked. This section points out issues in CT's trust model and discusses potential attack vectors resulting from unreliable monitors within the CT-enabled PKI.

A. The Trust Model of CT

CT enables *highly trustworthy* TLS server certificate services [5], [29], which ensure that any certificate acceptable to CT-compliant browsers is visible to its domain owner. The CT framework enforces two principles: (*a*) any centralized component (i.e., CA or log server) is inspected by others because such an entity might be benignly or maliciously faulty due to software flaws or attacks [4], [11], [12]; and (*b*) the correct behavior of a non-centralized component (i.e., auditor, browser, domain owner, or monitor) is ensured by redundancy or its own interest.

For each accepted TLS server certificate, the verification of multiple SCTs by a browser and corresponding audit paths by an auditor guarantees that any accepted certificate is visible in multiple logs, while the behavior of a log server is audited by redundant auditors. A domain owner acts as a self-monitor or employs the services of third-party monitors to detect potential bogus certificates. So the certificate signing services of a CA are inspected by redundant log servers, as the appended-only certificate records of a log are done by redundant auditors. Meanwhile, a browser verifies SCTs and audit paths (by acting as an auditor) for its own interest, and so does a domain owner (or self-monitor) detecting bogus certificates, because misbehavior of such a component would bring loss to itself.

However, *third-party monitors* that could be benignly or maliciously faulty similar to other components are not well-inspected in the CT framework. Although self-monitors are

[2]Cloudflare Monitoring, DigiCert Monitoring, EZMonitor, Hardenize and Report-URI provide only the subscription services and CT-Observatory was suspended in September 2018.

[3]Some monitors also use other name fields such as organization (O) and organizational unit (OU) as keywords.

[4]While CT-compliant browsers such as Chrome [20] and Safari [1] match only the SAN:dNSName filed to the visited domain, our experiments showed that the monitors also search the CN field of the certificates.

TABLE I: Services provided by third-party monitors on the Internet.

	URL	Service		Status	Certificate Signing
		Search	Subscription		
Censys	https://censys.io/	√		Running	
crt.sh	https://crt.sh/	√		Running	
Entrust Search	https://ui.ctsearch.entrust.com/ui/ctsearchui/	√		Running	√
Facebook Monitor	https://developers.facebook.com/tools/ct/	√	√	Running	
Google Monitor	https://transparencyreport.google.com/https/certificates/	√		Suspended	
SSLMate Spotter	https://sslmate.com/certspotter/	√	√	Running	
Cloudflare Monitoring	https://dash.cloudflare.com/		√	Running	√
DigiCert Monitoring	https://www.digicert.com/secure/		√	Running	√
EZMonitor	https://www.keytos.io/ezmonitor_overview.html		√	Running	
Hardenize	https://www.hardenize.com/		√	Running	
Report-URI	https://report-uri.com/account/		√	Running	
CT-Observatory	https://www.ct-observatory.org/	√		Suspended	

assumed in the original design, third-party monitors are more commonly used by ordinary domain owners in practice.

B. Attack Scenario in the CT-enabled PKI

In the CT-enabled PKI, the attacker aims to obtain a bogus certificate for the victim domain while remaining undetected by the domain owner. A potential attack consists of two steps: (1) *Obtaining bogus certificates:* The attacker could deceive the CAs that use domain validation (DV) to verify the domain ownership, e.g., by exploiting the vulnerabilities in the border gateway protocol (BGP) to hijack the traffic destined for the victim's domain [4], or utilizing DNS cache poisoning to fake the ownership of the victim's domain [11], [12]. And (2) *Concealing bogus certificates:* If a third-party monitor has design or implementation flaws that prevent it from returning certain certificates to users, the attacker could deliberately craft the bogus certificate and exploit the flaws to conceal it. For example, we found that Entrust Search (during our experiments) used a case-sensitive manner when looking up certificates for an inquired domain. Then, the attacker could create a bogus certificate whose SAN:dNSNames contain both upper- and lower-case letters (e.g., "WwW.fAceBOoK.cOm"), so that it would not be included in the result returned by Entrust Search when users inquire a domain with all lower-case letters (e.g., "www.facebook.com"). More potential exploitations are discussed in Appendix A.

C. The Threat Model

Third-party monitors in CT could be faulty or malicious, causing certain *relevant certificates missing* or *irrelevant certificates included* in the returned result. The misbehavior of benignly faulty monitors is likely rhythmic and repeatable. For example, a case-sensitive service never returns a certificate with a mixed-case SAN:dNSName for an inquired domain with all lower-case letters. On the other hand, the misbehavior of malicious monitors is commonly erratic and reproducible. They may return incorrect results once and then restore to the correct service, or respond with different results for the same domain in the split-view attacks [14], [24]. Nevertheless, while a single monitor may yield an incorrect result due to various reasons, a federation of monitors is very likely to return the complete set of relevant certificates.

We introduce the watcher component into the CT framework to ensure reliable certificate search services for third-party monitors. Such reliable services are necessary for CT as described above. The proposed watchers are mainly to detect the benign faults of monitors while providing a defense against maliciously faulty monitors to a certain extent. A watcher might also be faulty: (*a*) It could maliciously disclose monitor faults that do not exist. On one hand, there are no adverse effects due to the repeatability of benign misbehavior. On the other hand, irreproducible malicious behavior requires disclosure by multiple watchers and further manual confirmations. (*b*) Besides, it could detect the misbehavior of third-party monitors without disclosing it. However, redundant watchers on the Internet ensure that at least one correct watcher discloses the misbehavior.

IV. CT Watcher: Publicly Inspecting Third-party Monitors

This section presents the design of the proposed watchers, following the principles and assumptions of CT.

A. Design Goals and Challenges

A watcher is to audit the services of third-party monitors and detect any misbehavior that causes them to return incorrect results within the scope of its promised services (e.g., service delay). In this work, we focus on the search service, but it could be extended to the subscription service (as discussed in Section VII-B). Therefore, watchers are expected to achieve three design goals: (1) *effective misbehavior detection*, which requires the watchers to detect incorrect or abnormal behavior by actively probing and inspecting the monitors' services, (2) *scalable deployment*, which requires the operations of a

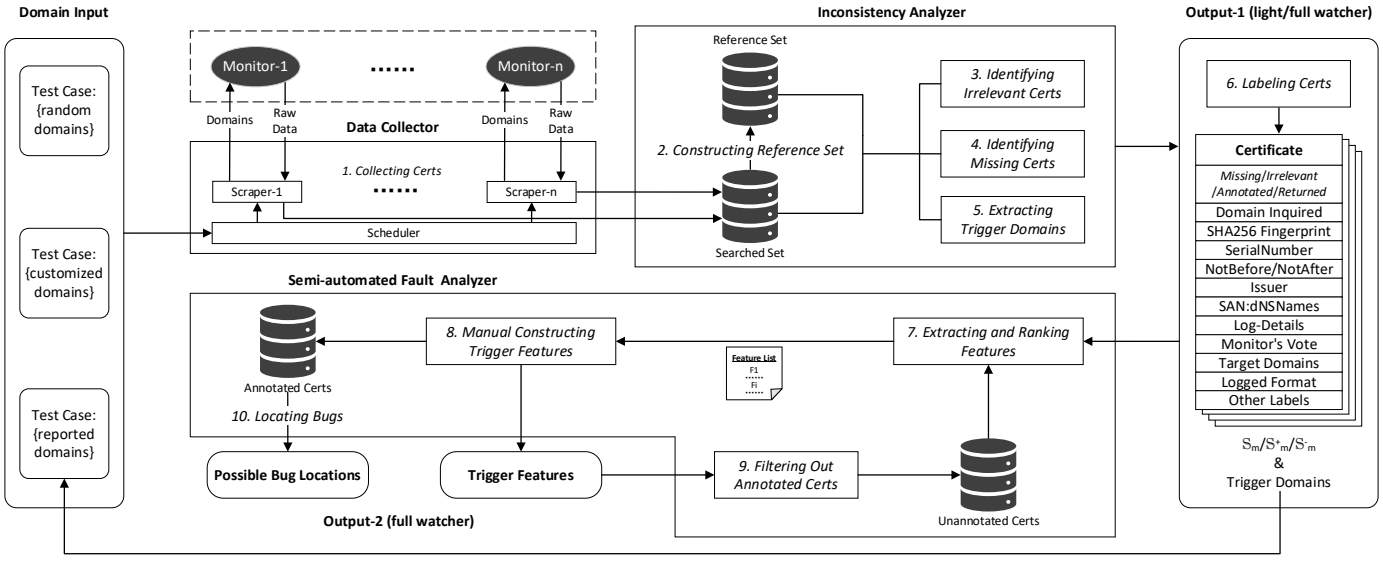


Fig. 2: The architecture and workflow of light/full watchers.

watcher to be automated and lightweight so that it can be continuously (or periodically) run by any interested party in the ecosystem (such as a domain owner), and (3) *(semi-)automated fault analysis*, which aims to uncover the possible causes behind the detected misbehavior and in some cases even identify the locations of the bugs. Such information provides useful insights for third-party monitors to review their configurations and implementations.

Designing watchers with these desirable goals faces non-trivial challenges. First, there lacks any ground truth or authoritative knowledge about the "correct" result that a monitor should return, i.e., for an inquired domain, the complete set of relevant certificates is *unknown*. Second, as it is impossible to predict the occurrence of a monitor's misbehavior, the watcher needs to audit and inspect all the domains. This requires a distributed design with many watchers cooperatively inspecting the entire space. Thus, the operational cost of a single watcher should be lightweight. Third, the monitors are largely diverse in terms of their query APIs, certificate sources, and search policies. Meanwhile, their internal processing is not open to the public. This requires non-trivial effort for automated certificate parsing and matching (discussed in Section V-A). Finally, a monitor's misbehavior may be caused by diverse reasons, including unintended flaws and malicious actions, which results in a large number of "abnormal" records (e.g., TABLE III shows 80K-633K inconsistent certificates per monitor). It makes manual analysis impossible while posing challenges to developing automated tools for fault analysis.

B. Assumptions

First, we assume a standardized interface for the inspected third-party monitors to output (*a*) the expected certificates for a inquired domain, and (*b*) a specific certificate for an inquired SHA256 fingerprint. Each output item contains basic-info (including the fields of CN, SAN:dNSName, Issuer, SerialNumber, NotBefore, NotAfter, etc.), cert-file (i.e., the PEM file), and log-details (including the name of a log server and the time the certificate was submitted).

We assume that third-party monitors publicly disclose the information about their quality of service (QoS): (*a*) *service delay*, the maximum delay promised by the monitor, from the time when a certificate is submitted to a monitored log until it appears in returned results; (*b*) *output limit*, the maximum number of certificates it can return for an inquiry; and (*c*) *log list*, the list of logs it monitors. This information assists in inconsistency analysis for watchers and helps domain owners select a suitable service. Note that any defects in these issues may result in missing expected certificates.

We design watchers following these assumptions. Although there is currently no uniform interface for monitors, we bridge the gap between these assumptions and real-world services through the implementation of middleware. Meanwhile, if the QoS information of some monitors is not disclosed, we deduce it by analyzing the results returned.

C. The Architecture of CT Watchers

A watcher searches for certificates for selected domains from inspected monitors. Then, it constructs a *reference set* for each domain based on all the returned results and analyzes the missing certificates and the irrelevant ones for each monitor.

A watcher typically runs continuously for N periods, and in each period completes the above steps using all certificates that have been collected. In the i-th period, the watcher analyzes and updates/outputs the inconsistent certificates of the inspected monitors in the first i periods. As more and more data are collected in the analysis (i.e., the value of i increases), the results output by the watcher are more accurate, see Section IV-D for details. Finally, after N periods, a *light watcher* (with inconsistency analysis only) concludes the inspection results of each monitor for each period and publishes its inconsistency with the reference set (if any) to the faulty monitors, a regulator, or even a public forum. Besides, a *full watcher* (with inconsistency and fault analysis) with sufficient resources additionally analyzes and reports possible benign failure causes. The architecture is shown in Fig. 2.

1) Inconsistency Analyzer: From each inspected monitor, a watcher periodically obtains a *searched set* of certificates issued for each inquired domain, and the time of conducting a search is denoted as *SearchedTime*. Due to possible faults, a searched set may contain irrelevant certificates and miss some expected ones. After the i-th period, the watcher constructs a *pre-reference set* as the deduplicated union of all searched sets from the previous i periods (but keeps only certificates that are valid at the SearchedTime), and a *reference set* is then obtained by removing all irrelevant certificates. In addition, through this merging process, the watcher knows the time when each certificate was submitted to a log for the first time, denoted as *SubmittedTime*, based on the returned log-details. For redundancy, the monitors are considered to monitor all the logs.

We identify irrelevant certificates by voting. A certificate returned by only a few monitors (e.g., ≤ 2 in our implementation and experiments; or the threshold depends on the total number of inspected monitors) is considered irrelevant.

For each inspected monitor, based on the reference set the watcher calculates an *irrelevant set* and a *missing set* for each period, any of which may be empty. A certificate that appears in the reference set but not in the monitor's searched set is considered a missing certificate. Conversely, it is considered an irrelevant certificate. Any domain that causes a monitor to return inconsistent results (i.e., the irrelevant set or the missing set is not empty) is identified as a *trigger domain*.

The missing certificates due to service delay, output limit, and unmonitored logs are further identified as these are considered "intentional" flaws in configurations. Specifically, (*a*) if the SubmittedTime of a missing certificate plus the monitor's service delay is earlier than the SearchedTime (as shown in the second rail of Fig. 3), it is counted as missed due to service delay, known as a *delayed certificate*; (*b*) if a domain's searched set reaches the monitor's output limit, and the reference set exceeds the limit, the missing certificates for this domain are considered to be caused by the output limit; and (*c*) if all logs recording a certificate are not in the monitor's log list, it is counted as missed due to unmonitored logs.

2) Semi-Automated Fault Analyzer: It is almost impossible to deduce the exact internal bugs of a monitor through external analysis alone. However, each bug leads to many similar misbehavior, which are usually rhythmic and repetitive. As a result, we merge the certificates for N periods as input of the fault analyzer, rather than analyzing each period separately, and then employ machine learning (ML) to automate clustering. Finally, the fault analyzer provides sample features for each monitor that can potentially trigger a bug, and/or identifies the location of the bug for further investigation.

The certificates are first automatically or manually labeled. Certificates in the missing/irrelevant set of any period are automatically labeled with "Missing"/"Irrelevant", while certificates missed due to service limitations (i.e., service delay, output limit, and unmonitored logs) are labeled with "Annotated". The rest are labeled with "Returned". We also manually attach some labels to each certificate (e.g., its lifetime and the number of its SCTs) to facilitate feature extraction. Then, the fault analyzer leverages a machine-learning model to extract and rank features. We focus on high-ranking features with discriminative power in classifying missing/irrelevant certificates and manually refine them into *trigger features* that show a strong correlation with possible bugs. The certificates with an identified trigger feature are then assigned an "Annotated" label. In this way, we eliminate the missing/irrelevant certificates gradually and reduce the burden of manual analysis in the next round. Finally, we repeat the above three steps until no similarity is observed for the remaining missing/irrelevant certificates.

We sample several missing certificates for each trigger feature and perform a few more black-box tests attempting to locate at which step in the workflow an error occurred. We first search the certificate by its SHA256 fingerprint or the precise subdomain. If it cannot be found by using the SHA256 fingerprint, the error may occur during the fetching or storing process. If it cannot be found by the precise subdomain, the error may occur during indexing. Otherwise, the error could occur when searching or returning the certificate. This process is known as *locating bugs.*

3) Domain Input: The inputs or test cases of watchers can be (*a*) *random domains*; (*b*) *customized domains*, domains of interest to the operators (e.g., their own domains or the ones in browsing history); or (*c*) *reported domains*, a collection of trigger domains reported to a public forum by watchers.

Light watchers tend to pick random and customized domains to inspect monitors for any misbehavior, as well as build the database of reported domains, while a full watcher usually selects reported domains to gather enough inconsistent certificates and then analyze the monitor faults. Moreover, more labels will be incorporated into the prototype implementation of full watchers and continuously improved. Thus, as the ecosystem is established, the analysis results by the full watchers become more and more accurate.

D. Impact from Service Delay

The service delay of monitors impacts the inconsistency analysis of watchers.

1) Tracking Time vs. Service Delay: Only after the maximum service delay (MSD) of inspected monitors, a newly-issued certificate is certainly retrieved by a watcher. So, only when the tracking time ($T = N * p$, p is the duration of each period) is long enough (i.e., $T > MSD$), the watcher can output accurate results for the first ($N - MSD/p$) periods.

Impact on reference sets. Continuous tracking for T length of time eliminates the deviation of a reference set due to the possible service delays of monitors. However, if T is too small (i.e., $T \leq MSD$), some expected certificates may not be retrieved within the tracking period, which will not appear in the reference set. If T is long enough (i.e., $T > MSD$), then in theory the expected certificates for the first ($N - MSD/p$) periods will all be discovered, which means that the reference set of these periods is theoretically accurate.

Impact on irrelevant sets. The voting mechanism is based on the assumption that irrelevant certificates are only returned by a few monitors. However, the monitor's ability to return newly issued certificates is closely related to its processing speed. Therefore, certificates submitted right before the $SearchedTime_N$ can only be returned by monitors that

TABLE II: Introduction of the actual implementation of monitors involved.

	Data Source		Cert. Info.†	Comparison Scope	Input vs. Result*			Expired Cert.	(Pre)Cert. Pair	API	Output Limit‡	
	Log List	Scan			Fingerprint	z.y.x	z.y.x w/ sub opt.				Web	API
Censys	47	√	B C L	CN SAN:dNSName	√	ab	-	optional	include	Python BigQuery¶	○	25000 ○
crt.sh	53	√	B C L	CN SAN O OU	√	a	ab	optional	optional	Postgresql	10000	○
Entrust Search	⊘	-	B L	CN SAN	-	a	ab	optional	include	-	5000	-
Facebook Monitor	⊘	-	B C	CN SAN:dNSName	-	ab	-	include	include	Graph	○	○
Google Monitor	⊘	-	B L	CN SAN:dNSName	√	ac	abc	include	include	-	○	-
SSLMate Spotter	56	-	B C	CN SAN:dNSName	-	a	ab	exclude	deduplicate	Json	○	○

√: the monitor supports this feature.
⊘: the monitor does not disclose its log list, so it's assumed to monitor all non-testing logs. †: *B* indicates basic-info, including CN, dNSName, SerialNumber, Issuer, NotBefore, NotAfter, SHA256 fingerprint. *C* indicates cert-file. *L* indicates the log-details.
*: "z.y.x" represents the domain user input, and "z.y.x w/ sub opt." means the query with the subdomains option. For a domain, the monitor returns any certificate binding (a) the domain inquired; (b) any subdomain name with wildcard or not; (c) the parent domain with wildcard.
¶: We contacted Censys for help and received an unlimited BigQuery database (censys-io) interface.
‡: The number represents the maximum number of the returned result set by the monitor; ○ means no explicit output limits.

perform well on timeliness, and they should not be considered irrelevant certificates even if they have few votes. Specifically, if the SubmittedTime of a certificate plus the MSD is later than the $SearchedTime_N$ (i.e., $SubmittedTime + MSD \geq SearchedTime_N$), then it should not be in the irrelevant set. In conclusion, if $T \leq MSD$, some irrelevant certificates may be counted in the reference set since most monitors did not return in time. If $T > MSD$, the judgment on irrelevant certificates in the first $(N - MSD/p)$ periods is accurate.

Impact on delayed certificates. A newly issued certificate but missing due to the monitor's mishandling, could be incorrectly classified as a delayed certificate. However, if the tracking time is long enough (i.e., $T > SD_m$, SD_m is the monitor m's service delay), certificates newly issued in the first $(N - SD_m/p)$ periods can be correctly classified by the watcher.

2) Certificate Lifetime vs. Service Delay: A considerable delay in the monitor's service creates a substantial attack window, potentially causing the omission of even short-lived certificates. Consequently, the voting mechanism becomes ineffective in identifying irrelevant certificates among these short-lived ones. Specifically, certificates with a very brief lifespan (less than MSD) that receive minimal votes are not considered irrelevant by the watcher.

V. Implementation

In this section, we present the implementation details of a watcher. The prototype supports 6 actively running monitors with public search services.

A. Monitor Investigation

Among the 6 monitors, SSLMate Spotter and crt.sh disclosed partial source code to demonstrate their search [9] and subscription [49] services, from which we obtained basic information about how the monitors fetch and process certificates. We also explored the monitors' official websites and some CT forums, exchanged emails with monitors' development teams, and conducted several black-box tests to extract information about each monitor's implementation.

Service information. The following service information is useful in the inconsistency analysis, to distinguish inconsistent certificates due to faults and "intentional flaws". Only some monitors disclose partial information publicly, and we infer other configurations through the returned results.

1) *Service delay*. None of the monitors disclosed their service delays, and we estimated monitors' service delays from long-term tracking experiments. The experimental results show that the service delay of Censys is 15 days, crt.sh is 6 days, Facebook Monitor is 3 days, Google Monitor and SSLMate Spotter is 2 days, and Entrust Search is 34 days. Moreover, we dynamically set the service delay of Entrust Search and evaluate its impact on a watcher (see Section VI-B for details).

2) *Output limit*. Entrust Search provides services with an output limit of 5,000. For each of the other monitors, we choose an interface without output limits according to the monitor's claim.

3) *Log list*. Among 6 inspected monitors, only Censys, crt.sh, and SSLMate Spotter disclose the log list. If a monitor does not disclose its log list, we assume that it monitors all non-testing logs.

Differences in implementations. This subsection details the differences between monitors, which influence the design of unique processing engines (i.e., scraper in Section V-B) used to collect data, as shown in TABLE II.

1) Certificate source. All monitors fetch certificates from public log servers. However, some certificates in Censys and crt.sh are obtained through active scans or from test logs, which are beyond the scope of CT Watcher.

2) Returned certificate information. All monitors return *basic-info* of a certificate. Besides, some monitors return *log-details* and/or the *cert-file* file of the certificate.

3) Search policy. We compare search policies and summarize four main differences: (*a*) "comparison scope" defines the format of search items; (*b*) "input *vs.* result" describes the matched certificates returned by a monitor for an input; (*c*) "expired cert." indicates whether the returned results contain expired certificates; and (*d*) "(pre)cert. pair" indicates whether the returned result deduplicates the (pre)certificate pairs. Specifically, with the exception of SSLMate Spotter, none of the other monitors deduplicated the returned (pre)certificate pairs.

Moreover, Facebook Monitor and Google Monitor search results contain a large number of expired certificates. Additionally, crt.sh also returns some unexpected certificates by matching O and OU. It is necessary for scrapers to deduplicate and filter certificates for these cases.

4) Query API. Censys and crt.sh are accessible via a database, Facebook Monitor and SSLMate Spotter can be accessed via HTTPS requests, while Google Monitor and Entrust Search do not provide any API. Therefore, scrapers have to collect data through various APIs. In addition, all monitors support search by domain names, while Censys,crt.sh, and Google Monitor additionally accept SHA256 fingerprint as the search keyword.

B. Data Collection

This module is designed to obtain standardized data from third-party monitors. The engine consists of a *scheduler* and multiple *scrapers*, each specific to a monitor. Scraper excludes differences in results between monitors due to search policies, making the obtained data available for differential analysis. The scheduler loads the domains from the selected test cases and creates tasks to be submitted to each monitor. Moreover, it ensures that the queries about the same domain are submitted to all monitors synchronously to reduce the impact of service delay on the returned results.

Scrapers. For each monitor, we implement a unique scraper (see Appendix B for details) that prepares one or multiple queries based on its search policy and its search syntax of API. We uniform the format of expected search results. For each domain inquired, the monitor *m*'s scraper fetches *raw data* from *m* and then returns the searched set of *m*, which will be directly compared with each other. Each certificate in the searched set is uniquely identified by a quadruple (SerialNumber, Issuer, NotBefore, and NotAfter), which is used to deduplicate (pre)certificate pairs. Additionally, log-details are used to determine if a certificate is missed due to service delay or unmonitored logs. Therefore, this quadruple plus the log-details is the minimum certificate information required by the *light watcher*.

C. Inconsistency Analyzer

Constructing the reference sets. To address the possible lack of newly issued certificates due to monitors' service delays, we construct the reference set for a domain d in the i-th search period ($\mathbb{R}_d^i$) in three steps. Let us denote the searched set returned in the i-th period by a monitor m for a domain d as $\mathbb{S}_{d,m}^i$. In the first step, we combine the searched sets of all the monitors returned in each of the N periods, to generate a total set $\mathbb{T}_d$, as shown in Equation (1).

$$\mathbb{T}_d = \bigcup_{i\in[1,N], m\in\mathbb{M}} \mathbb{S}_{d,m}^i \tag{1}$$

$$\tilde{\mathbb{R}}_d^i = \mathbb{T}_d - \mathbb{U}_d^i - \mathbb{E}_d^i \tag{2}$$

$$\mathbb{R}_d^i = \tilde{\mathbb{R}}_d^i - \bigcup_{m\in\mathbb{M}} \mathbb{S}_{d,m}^{+\ i} \tag{3}$$

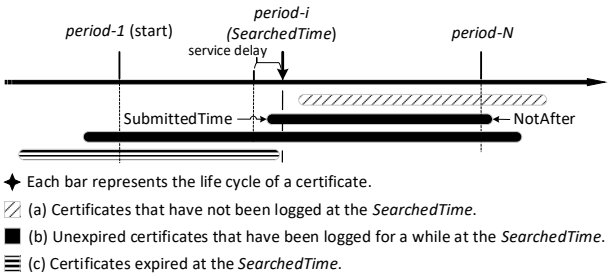


Fig. 3: Certificates in three states at the SearchedTime.

To construct the pre-reference set in the i-th period ($\tilde{\mathbb{R}}_d^i$), we need to remove the certificates that are unlogged at the i-th period ($\mathbb{U}_d^i$) and the ones expired before the i-th period ($\mathbb{E}_d^i$), following Equation (2). What's more, Fig. 3 shows the lifetime of these two types of certificates. Finally, the watcher uses the method in the next paragraph to remove the irrelevant certificates to generate the reference set of the domain d for the i-th period ($\mathbb{R}_d^i$), following Equation (3).

Identifying irrelevant certificates. The watcher counts the votes for each certificate. If the monitor m returns a certificate for the domain inquired d within the N periods of tracking (i.e., $certificate \in \bigcup_{i\in[1,N]} \mathbb{S}_{d,m}^i$), then it votes for this certificate. If a certificate gets only a few (i.e., ≤ 2) votes, the watcher will identify it as irrelevant. It is worth noting that some newly issued certificates (i.e., $SubmittedTime + MSD \geq SearchedTime_N$) and short-lived certificates (i.e., $lifetime \leq MSD$) are excluded because the processing speed of monitors heavily affects the voting results for these certificates. Ultimately, for each monitor m, the set of irrelevant certificates for the i-th period of domain d is $\mathbb{S}_{d,m}^{+\ i} = \mathbb{S}_{d,m}^i - \mathbb{R}_d^i$.

Identifying missing certificates. For each monitor m, the watcher calculates a set of missing certificates for a domain d in the i-th period, which we denote as $\mathbb{S}_{d,m}^{-\ i}$. This set can be directly computed as $\mathbb{R}_d^i - \mathbb{S}_{d,m}^i$. The certificates missed due to service limitations are identified according to the method in Section IV-C1, and the set containing these certificates is denoted as $\hat{\mathbb{S}}_{d,m}^{-\ i}$.

Extracting trigger domains and constructing the output. We extract the domains that trigger inconsistent certificates, building a set of trigger domains for each monitor. Finally, the continuously running watcher updates/outputs the $Result$ every period, which contains the query status (i.e., $[SearchedTime_i, \mathbb{S}_{d,m}^i, \mathbb{S}_{d,m}^{+\ i}, \mathbb{S}_{d,m}^{-\ i}, \hat{\mathbb{S}}_{d,m}^{-\ i}]$) of each trigger domain for each monitor in the past few periods.

D. Semi-Automated Fault Analyzer

Labeling certificates. The fault analyzer takes all collected certificates (i.e., $\mathbb{T} = \bigcup_{d\in domains} \mathbb{T}_d$) as input to classifiers. In the monitor m's classifiers, certificates in $\mathbb{S}_m^+$ (i.e., $\bigcup_{d\in domains}^{i\in[1,N]} \mathbb{S}_{d,m}^{+\ i}$) are labelled as "Irrelevant", and those in $\mathbb{S}_m^-$ (i.e., $\bigcup_{d\in domains}^{i\in[1,N]} (\mathbb{S}_{d,m}^{-\ i} - \hat{\mathbb{S}}_{d,m}^{-\ i})$) labelled as "Missing".

Each certificate is labeled with the following three aspects (see TABLE IX in Appendix C for details): (*a*) *certificate nature*: e.g., issuer and lifetime; (*b*) *submission characteristics*:

e.g., number of SCTs and *logged format* of the certificate in logs (i.e., 'Pre' for precertificate, 'Final' for final certificate, and 'Both' for both precertificate and final certificate); (*c*) *information related to domain inquired*: e.g., number of *target domains* (i.e., domains matched by the domain inquired in the certificate) and number of *uncorrelated domains* (i.e., domains other than the target domains in the certificate).

Extracting and ranking features. Using labeled certificates, we extracted 778 features from these certificates and trained a random forest (RF) model with 10 estimators for each monitor to predict if a certificate with specific patterns would be returned in the search results. Also, the RF models output ranked features based on their importance in prediction, which provides insight to guide manual analysis.

Manually constructing trigger features. We focus on the top 5 features for each monitor, which have a strong guiding role, and construct possible trigger features. For example, the most salient feature for the Facebook Monitor is the "average size of reference sets". With this clue, we focused on domains with mass certificates. We found that Facebook Monitor returns search results by page. If the result took multiple pages, missing or duplicate certificates occurred. We speculated that this was caused by gaps or overlaps between pages and constructed trigger feature F9 in TABLE V.

The fault analyzer determines whether the constructed feature is a real trigger feature, which is defined in two types. Specifically, some features that certificates match cause them to be missed with high probability, which can be determined by Measure 1. In addition, certificates matching certain features are not missed with high probability, but their proportion in the irrelevant/missing set is significantly higher than the normal proportion, which also shows that these features have a strong correlation with possible bugs and can be determined by Measure 2.

Measure 1. For a monitor m and a feature F_x, we compute the ratio between the number of inconsistent certificates of m matching F_x and the number of all collected certificates matching F_x, as in Formula (4). There is a large ratio indicating there is a strong correlation between feature F_x and monitor m's inconsistency on search results.[5]

$$P = \frac{\#\text{ of cert.} \in \mathbb{S}_m^+/\mathbb{S}_m^- \text{ matching } F_x}{\#\text{ of cert.} \in \mathbb{T} \text{ matching } F_x} \tag{4}$$

Measure 2. The Cramer's V for chi-square goodness-of-fit test [56], [43] is adopted. First, we calculate the number of inconsistent certificates matching or not matching F_x as O_1 and O_2. Second, we hypothesize that F_x is independent of the inconsistent. Then, we calculate the expected value of inconsistent certificates matching or not matching F_x as E_1 and E_2, where n is the number of inconsistent certificates and p_i ($i \in [1, 2]$) is the proportion of certificates matching or not matching F_x in $\mathbb{T}$. Third, we calculate the statistic chi-square random variable, denoted as X^2, and a value of Cramer's V, denoted as V in Formula (5). If the $V \geq 0.5$, it means that F_x has a high association with the inconsistency [8].

[5]To prevent the defined features from being too deviant, the threshold in Measure 1 should not be too small, so we set it to 0.9 according to the experimental data.

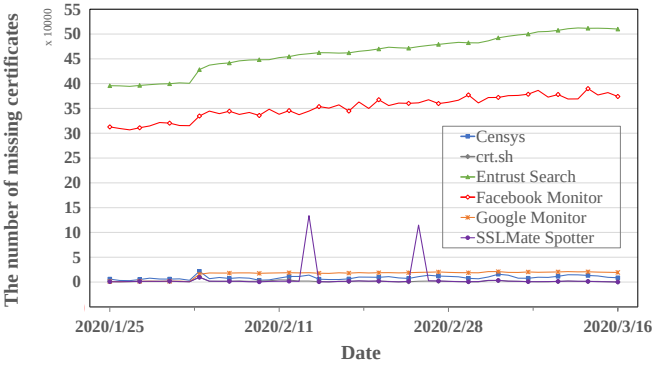


Fig. 4: Overview of missing certificates.

$$E_i = n * p_i; \qquad X^2 = \sum_{i=1}^{2} \frac{(O_i - E_i)^2}{E_i}; \qquad V = \sqrt{\frac{X^2}{n}} \tag{5}$$

VI. Experimental Evaluation and Analysis

Our experiments consist of two parts: (*a*) a trial operation to evaluate the effectiveness of the watcher designs; and (*b*) confirmations to verify the flaws identified.

We randomly select 1,000 domains from Alexa Top-1K, Top-1K~10K, Top-10K~100K, and Top-100K~1M separately, 4,000 in total. Then, we conducted a 52-day trial operation of the watcher prototype from January 25 to March 16, 2020. Finally, the watcher took 3,996 domains as inputs and eliminated 4 super domains (i.e., "amazonaws.com", "zendesk.com", "azure.com", "netflix.com") due to difficulty in completing the inquiry within 24 hours.

Approximately 5M certificates were obtained each day during the trial operation, and 267.6M were obtained in 52 days. This data collection contains many duplicate certificates obtained from different monitors on different days, together with precertificates and final certificates. After deduplication (i.e., only one of the corresponding precertificate and the final certificate remains), there were 964,050 unique certificates.

A. Inconsistency of Returned Results

TABLE III shows an overview of inconsistent certificates for 6 monitors. Some monitors returned irrelevant certificates. During the trial operation, there were 665 certificates with votes ≤ 2. Among them, 570 certificates were issued during the last two days of the trial operation and 1 certificate was only valid for one day, which were not counted as irrelevant certificates, as explained in Section IV-D1. In addition, 42 certificates were only returned by Facebook Monitor, which was finally confirmed to have a time parsing error, as described in Section VI-C (F5). In addition, 52 email certificates were only returned by Censys and/or Entrust Search.

Fig.4 shows the number of missing certificates for each monitor in the trial operation. SSLMate Spotter encountered an internal service error when querying "ondemand.com" on February 14th and "cisco.com" on February 25th, making two peak values appear. A mass of short-lived certificates (lifetime

TABLE III: Inconsistent certificates.

	Censys	crt.sh	Entrust Search	Facebook Monitor	Google Monitor	SSLMate Spotter
$\mathbb{S}^+$	-	52	5	42	-	-
$\mathbb{S}^-$	206,037	80,841	621,520	633,605	95,527	310,078
SD	203,030	80,841	76,999	38,862	75,258	65,365
OL	-	-	466,828	-	-	-
LL	11	-	-	-	-	-
IE	-	-	-	-	-	244,713
SB	2,973	-	65,447	594,737	19,939	-
UC	23	-	12,246	6	330	-

SD, OL, and LL denote the number of certificates that a monitor missed due to service delay, output limit, and log list, respectively. Similarly, IE, SB, and UC represent the number of missing certificates due to informed errors, service bugs, and unknown causes.

< 91 days) were issued for "wixsite.com" on February 3rd, but none of the monitors could process them in time, making the number of missing certificates surge. Furthermore, flaws in the handling of short-lived certificates by the Google Monitor led to the inability to locate these certificates at a later time. Besides, the number of Entrust Search's missing certificates continues to increase, which indicates a certificate backlog problem.

Our experiments showed that monitors rarely returned irrelevant certificates, but missed 8.4% - 65.7% of certificates during the trial operation. Specifically, there were 206,037 certificates (out of 964,050 certificates) for Censys that could not be guaranteed to always be returned correctly, 80,841 for crt.sh, 621,520 for Entrust Search, 633,605 for Facebook Monitor, 95,527 for Google Monitor and 310,078 for SSLMate Spotter. Moreover, for Censys there were 3,007 certificates that were always missed, 0 for crt.sh and SSLMate Spotter, 517,837 for Entrust Search, 384,560 for Facebook Monitor, and 19,939 for Google Monitor.[6]

The investigation found that the output limit of Entrust Search was 5,000. Based on this limitation, 466,828 missing certificates for 22 domains were annotated. In addition, Censys did not monitor Google Argon 2022 at the time resulting in 11 missed certificates. What's more, the empirical setting of service delays also annotated a large number of delayed certificates, see Section VI-B for details. Eventually, Censys was left with 2,996 missing certificates, 77,693 for Entrust Search, 594,743 for Facebook Monitor, 20,269 for Google Monitor, 244,713 for SSLMate Spotter (the code "internal_error" was used to inform the error), and 0 for crt.sh.

B. Service Delays

In practice, the delay for a bogus certificate from being trusted to being detected is equal to the log server's merge delay plus the monitor's service delay. Both Meiklejohn et al.'s survey [34] and our measurements indicate that almost all log servers merge submitted certificates within a few minutes, see Appendix D for details. That is, the main delay of CT comes from the service delay of monitors.

However, no monitor currently discloses its service delay. Therefore, we made a statistic on about 219.8K newly issued

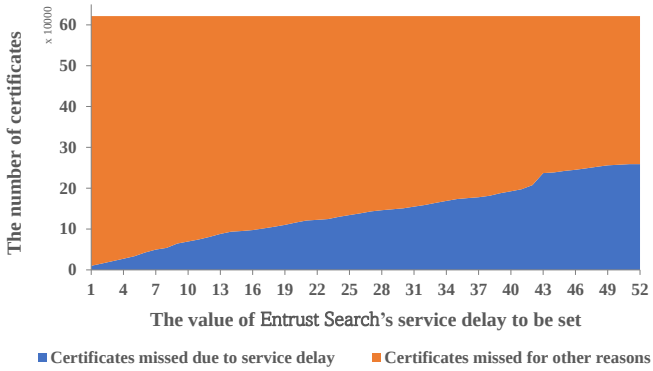


Fig. 5: The number of Entrust Search's missing certificates classified as delayed certificates under different settings of service delay.

certificates during the trial operation. Certificates that are issued during the experiment and returned correctly after n days are considered to be missed due to service delays, and those not returned are considered to be caused by service bugs. The results showed that almost all (over 99.99%) missing certificates for Censys due to service delay were found within 15 days, 6 days for crt.sh, 4 days for Facebook Monitor, 2 days for Google Monitor and SSLMate Spotter. Besides, the timeliness of Entrust Search was relatively poor, only about 32.9K newly issued certificates were found within 52 days, and 99% of them were found within 34 days. Even some certificates were not returned properly until 51 days after they were submitted, which means that 52 days of tracking is not enough to measure Entrust Search's actual service delays[7].

When the service delay is set to different values, the result of delayed certificates filtered by the watcher is different. If the service delay is set too small, only a part of the delayed certificates can be filtered out. Instead, if the service delay is set too large, some certificates missed due to service bugs will be wrongly classified as delayed certificates. Fig. 5 shows the related situation of Entrust Search: as the set service delay increases from 1 to 52, the proportion of missing certificates identified as delayed certificates rises from 1.6% to 41.6%.

C. Identified Faults

The RF classifiers reach a high test accuracy (0.912 for Entrust Search, > 0.994 for others) and an F-1 score between 0.62 and 0.982 for distinguishing the missing certificates from others, as shown in TABLE IV. The models show a high accuracy as the datasets are extremely imbalanced. Based on the high-ranked features, we manually construct 4 confirmed trigger features (i.e., F2, F4, F9, and F11), which filter out 83.0% to 99.8% of the missing certificates of monitors, see Appendix C for details.

Through the similarity analysis of missing certificates, we listed the features triggering certificate misses and confirmed multiple monitor bugs, as shown in TABLE V. We explained the causes of almost all the certificates missed from Censys,

[6]Delayed certificates and some missing certificates were still returned correctly in several periods of the trial operation.

[7]Entrust Search suspended its service for a few days on March 17th, making longer tracking impossible.

TABLE IV: Test F-1 scores of the RF classifiers for monitors and proportion of certificates annotated with trigger features constructed from high-ranked feature guidelines.

	Censys	Entrust Search	Facebook Monitor	Google Monitor
F-1	0.976	0.798	0.620	0.982
Annotate Rate	96.6%	83.0%	99.8%	98.2%

crt.sh, SSLMate Spotter, Google Monitor and Facebook Monitor, except for 330 from Google Monitor, 23 from Censys and 6 from Facebook Monitor for unknown reasons. Besides, we explained about 98% of the missing certificates of Entrust Search. Then, we categorized the possible bugs of monitors, each corresponding to a feature in the list.

⋆ *Problems in Certificate Collection*

#F1: Entrust Search only monitors Google-operated logs. It could not find all the 35 certificates that were not submitted to any Google-operated logs.

#F2: Entrust Search is prone to miss certificates on the busiest logs during fetching/storing. Among the 19 Google-operated logs at the time, the 4 logs with the fastest growth (i.e., Google Rocketeer, Google Pilot, Google Xenon 2020, Google Argon 2020)[8] were defined as busiest-Google-logs and other 15 Google-operated logs were defined as non-busiest-Google-logs. After excluding missing certificates that identified the cause, Entrust Search still missed 77,693 certificates, 64,475 of which were not logged in non-busiest-Google-logs, accounting for 83%. Since Entrust Search monitors only Google-operated logs, it missing certificates are from the busiest-Google-logs. The expected proportion of certificates with this feature is 40.6%. The calculated Cramer's value V is 0.863.

⋆ *Problems in Certificate Indexing*

#F3: Facebook Monitor and Google Monitor incorrectly indexed redacted certificates.[9] Both of them missed the only 2 redacted certificates [53] ("?.delta.com" and "?.?.salesforce.com"). Section VI-D provides further analysis.

#F4: Censys may incorrectly parse a certificate with vast characters in its SAN:dNSNames. There were 3178 certificates with a large number of characters (i.e., the total number of characters of SAN:dNSNames in the certificate is greater than 8,830) in the trial operation, of which 2895 (91.1%) were tagged as "unparsable" and provided with only the SHA256 fingerprint in Censys' web service.

#F5: Facebook Monitor incorrectly parsed the validity period of some certificates. We found that Facebook Monitor returned a different NotBefore or NotAfter for all 42 certificates matching F5 than the other monitors. If the time happens to be between 2:00 and 3:00 on the day when daylight saving time starts, then NotBefore or NotAfter will be +1 hour.

[8]During the trial operation, the entry growth of CT reached 13.4M per day, of which the daily growth of Google Xenon 2020 was about 3M (accounting for 22.4%), Google Argon 2020 was 2.7M (accounting for 20.4%), Google Rocketeer was 1.5M (accounting for 10.8%), and Google Pilot was 1.4M (accounting for 10.6%).

[9]To prevent privacy leakage, the redacted certificates use '?' to replace the privacy-carrying part in domains to prevent privacy leaks.

TABLE V: Manually constructed trigger features.

No.	Trigger Feature
F1	It does not submit to Google-operated logs.
F2	It only submitted to the busiest logs.
F3	Its *target domains* contain '?'.
F4	Its SAN:dNSNames contain over 8,830 characters.
F5	Its NotBefore or NotAfter is in a special time interval.
F6	It has only one *target domain*.
F7	Its *target domains* contain '_' and its *logged format* is 'Pre'.
F8	The *domain inquired* is an IDN-ccTLD.
F9	The search results of the *domain inquired* need to be returned on multiple segments.
F10	Its *target domains* are mixed cases.
F11	Its lifetime is less than 91 days.

#F6: Facebook Monitor may omit to mark certificates with a small probability. After excluding missing certificates that identified the cause, there were still 1,297 missing certificates. Among them, 1,291 had only one target domain, accounting for 99.5% However, the expected proportion of certificates with F6 is 58.5%. The calculated Cramer's value V is 0.833. See Section VI-D for further analysis.

#F7: Google Monitor may have an error in parsing the precertificate with underscores in its domains. During the preparation phase, we found 165 missing certificates of Google Monitor with F7, which were not included in the 52-day trial operation. We submitted this bug to Google, and they stated that there was a parsing error about some precertificates.

⋆ *Problems in Certificate Look-up or Output*

#F8: Entrust Search and Facebook Monitor prohibited queries with some internationalized domain name (IDN). For "xn--b1amahh6b.xn--p1ai" (2 certificates binding it), the only IDN with an internationalized country code top-level domain (IDN-ccTLD), Entrust Search and Facebook Monitor did not return any results, see Section VI-D.

#F9: Certificates were duplicated or missed when Facebook Monitor paged returned data. The search results of 122 domains were returned on multiple pages. Facebook Monitor returned incomplete results for 116 of them, accounting for 95.1%, see Section VI-D for further analysis. A total of 592,902 missing certificates were involved in this feature.[10]

#F10: Retrieval of Censys and Entrust Search is case sensitive. We discovered 1,428 certificates with mixed-case target domains. Entrust Search could not find all 1,428, so its search was case-sensitive. Differently, Censys could not find 17 whose CN was not a target domain since Censys was only case-insensitive for "parsed.subject.common_name" while case-sensitive for other 3 domain-related fields (i.e., "parsed.names", "parsed._expanded_names" and "parsed. extensions.subject_alt_name.dns_names").

#F11: There may be a problem with Google Monitor's outputs for short-lived certificates. Google Monitor has a total of 20,269 missing certificates, of which 19,901 have a lifetime

[10]The page-turning error causes certificates to be randomly missed, making them indistinguishable from certificates that were missed for other reasons. Therefore, we annotate all 592,902 certificates with this error.

TABLE VI: The comparison of IDN query.

	UniCode		Punycode	
	IDN*	IDN-ccTLD§	IDN†	IDN-ccTLD‡
Censys	⊘	⊘	√	√
crt.sh	⊘	⊘	√	√
Entrust Search	⊘	⊘	√	⊘
Facebook Monitor	⊘	⊘	√	⊘
Google Monitor	⊘	⊘	√	√
SSLMate Spotter	√	√	√	√

⊘ means the monitor does not support the query.
*: E.g., я.wiki; §: E.g., увики.рф;
†: E.g., xn--41a.wiki; ‡: E.g., xn--b1amahh6b.xn--p1ai.

of less than 91 days, accounting for 98.2%. However, the expected proportion of such certificates is 22.5%. The calculated Cramer's V is 1.812, which is greater than 1 due to unequal expected probabilities of certificates with and without F11. The details can be found in Section VI-D.

D. The Confirmed Bugs and Limitations

Regarding all the identified issues, we proceeded to analyze some other certificates (or domains) on the Internet that matched the same features, aiming to confirm if they were also being mishandled. In addition, for some specific issues, we also conducted several separate experiments to determine the underlying causes or find more related bugs. This subsection focuses on separate experiments from April to December 2020 involving a total of 7,295,986 unique certificates.

Redacted Certificate. We conducted experiments on 42,269 redacted certificates belonging to 1,898 domains. These certificates were used to explain the problem of Facebook Monitor and Google Monitor about F3. We found that (*a*) if all SAN:dNSNames in a certificate were redacted, it could not be found; (*b*) if part of SAN:dNSNames were redacted, it could be found by unredacted SAN:dNSNames. This may result from a mistake in monitor setting or matching indices.

Mark Error. 538 of the 1,291 missing certificates mentioned in Section VI-C (F6) contained not only the target domains but also some uncorrelated domains. The bug locating found that they could be found through uncorrelated domains[11] rather than precise subdomains. This may be caused by Facebook Monitor's omission to mark the index. When a certificate has only one target domain, it is more prone to being missed.

IDN. Inspired by Section VI-C (F8), we further experimented to explore the strategies of 6 monitors in querying IDNs. We used Unicode and Punycode to query for IDN-ccTLD and IDN with ordinary TLD respectively. As a result, only SSLMate Spotter could recognize the Unicode encoding of IDN. In addition, Facebook Monitor and Entrust Search judged an IDN-ccTLD as an invalid domain, as TABLE VI.

Page-turning Error. To confirm the bug about F9 in Section VI-C, we set the page size to 1,000 and 5,000 to synchronize multiple queries "uol.com.br", a domain that at the time had 4,419 relevant certificates bound to it. Facebook Monitor used a cursor item to locate a page. When the page size was 1,000, it took 5 pages to return the results, and both the cursors and the number of results varied. However, when the page size was 5,000, the number of results was steady. We submitted this bug to Facebook and it was fixed, but Facebook Monitor just set an output limit of 10,000.[12]

Output Limitation. We also found uninformed output limits by analyzing 4 super domains, see Appendix E for details.

1) *Google Monitor*. It only returns a part of certificates for super domains. Moreover, the upper limit is smaller for the domains whose relevant certificates were mostly short-lived (e.g., "wixsite.com"). But these missing certificates can be found when searching by precise subdomain, which means that the error is likely in the look-up stage.

First, we analyzed the returning order of certificates. For a certificate, we extracted the effective second-level domains (e2LDs) of target domains,[13] and defined the smallest one in lexicographical order (case-insensitive and numbers are smaller than letters) as smallest-e2LD. Google Monitor returned the results in the following order, (*i*) certificates binding the parent domain with wildcard; (*ii*) certificates binding the domain inquired with wildcard or not; (*iii*) certificates according to the lexicographical order of their smallest-e2LDs. There were also a few certificates returned out of order.

Then, we analyzed the output limit. Since Google Monitor returned unexpired and expired certificates by default, the next analysis included expired certificates. We again obtained the relevant certificates of 7 domains by Google Monitor (see TABLE XII in Appendix E), and found that it could only return the first part of certificates in lexicographical order. For example, when querying for "azure.com", only certificates whose smallest-e2LD starting with 'c' or lower in lexicographical order are returned.

2) Facebook Monitor. After Facebook Monitor set the output limit of 10,000, its search service for some domains (e.g., "microsoft.com") did not even return any valid certificate. The reason is that its Graph API returns the earlier certificate first, resulting in all returned certificates being expired.

Bad Tag. Censys sets bad tags for some certificates. First, it was found that some certificates in the web service were tagged as "unexpired" in the *Tags* field, but tagged as "Expired Leaf" in the *Browser Trust* field.[14] This bug was reported to Censys and confirmed. Second, we found that Censys had an issue with misjudging the validity period of certificates. It may tag an unexpired certificate as "expired", and vice versa. On June 22, 2020, we searched for certificates tagged "expired" with an expiration date after June 23, 2020, and found 66,245 certificates when there should have been 0 certificates.

Unknown Causes. There were still some missing certificates with undiscovered similarities that were not interpreted. Since many possibilities could lead to inconsistencies, some of which may occur randomly, it was very difficult to interpret them all. We did further bug locating on Censys which could search for certificates via SHA256 fingerprint.

[11] Facebook Monitor does not accept SHA256 fingerprint as the search keyword. Thus, instead of querying specific certificates via SHA256 fingerprint, we query uncorrelated domains in the certificate to complete the bug locating.

[12] Facebook Monitor didn't set output limits until we submitted the bug.

[13] For example, "msm.live.com" is the e2LD of "rps.msm.live.com".

[14] When interacting with Censys using the BigQuery interface and filtering expired certificates by the "unexpired" tag, these certificates were not returned.

TABLE VII: The average overhead of processing a domain.

	Downloads	Storage	Time	Cost
Censys	0.16MB	0.19MB	-†	Free/$0.04‡
crt.sh	0.23MB	0.26MB	5.21s	Free
Entrust Search	0.12MB	0.28MB	9.91s	Free
Facebook Monitor	0.16MB	0.28MB	14.1s	Free
Google Monitor	0.36MB	0.41MB	79.7s	Free
SSLMate Spotter	0.16MB	0.19MB	87.98s	Free/$0.002§
Watcher	1.19MB	1.97MB	163s	△

†: We use the BigQuery interface that Censys only provides to researchers, so its data does not have reference value;
‡: If 250 free queries per month are exceeded, Censys charges monthly (or yearly), and it costs $0.04 to search for a domain under full load. §: If 10 free queries per hour are exceeded, SSLMate Spotter charges monthly (or yearly), and it costs about $0.002 to search for a domain under full load.
△: The fee paid by the operator is related to the number of domains analyzed and the search period set.

For Censys' 23 unexplained certificates, 8 of them could not be found even when searched using their SHA256 fingerprints. This issue might arise due to Censys failing to fetch or store these certificates. The 31-day tracking of Censys' backlog has revealed that it has the issue of incomplete certificate inquiry for many logs (see Fig. 6(a) in Appendix E). In addition, 9 unannotated certificates cannot be found even when searched by their precise subdomains, possibly due to indexing errors. In our further investigation, we found that none of them has the "parsed.names" field. Additionally, the remaining 330 certificates of Google Monitor were only missed for one day during the 52 days of tracking, likely due to accidental service shocks.

E. Operating Cost of Light Watchers

Based on the trial operation, we estimate the average cost of a watcher to process a domain per search period (see TABLE VII for an overview). For 3,996 tracked domains, each domain contains about 240 relevant certificates on average, requiring about 1.19MB of data to download. In addition to the directly downloaded raw data, the watcher also constructs the *searched set*, the *reference set*, and the *irrelevant/missing set*. In general, a domain requires 1.97MB of storage on average.

We measured the time taken by each monitor to obtain certificates of domains selected using 5Mbps network bandwidth. Due to the design of synchronous execution, the average time for the watcher to search for a domain is about 163 seconds.

The 6 selected monitors are basically free, except for Censys and SSLMate Spotter, which charge for a large number of queries. When the operator needs to search more than 250 times a month (i.e., watching over 8 domains per day) , Censys charges at least $0.04 for each search. In addition, since SSLMate Spotter returns up to 100 certificates per search, an average of 3 searches is required for each domain. If the operator wants to perform more than 10 searches per hour (i.e., watching over 80 domains per day) through SSLMate Spotter, it needs to pay at least $0.002 for each search.

F. Disclosures and Responses

SSLMate Spotter performed relatively well, and no bugs were found, so we did not issue a report. Entrust Search did not respond to the bugs reported on September 17, 2021.

Although crt.sh performed well during the trial operation, our testing in May 2022 still found some issues. crt.sh only returned some of the top-ranked certificates, even for some domains that did not hit the output limit. For example, for "taobao.com", it only returned 80 certificates issued before September 7, 2021, which was about 140 fewer than expected. We reported the issue on May 23, 2022, but crt.sh did not provide a convincing response.

We reported the identified bugs to Censys. The bad tags bug (reported on January 16, 2020) was confirmed by them, but we have not received any response to the other bugs (reported on September 16, 2021).

For Facebook Monitor, we reported the page-turning error on December 4, 2019, and Facebook fixed it on September 28, 2020. We then reported additional findings on May 20, 2022, and received a response from Facebook Monitor, stating that it had been referred to the concerned team.

Google Monitor confirmed the issue with parsing errors for some precertificates, and fixed it on November 28, 2019. But Google Monitor stated that it was not a "monitor" as defined in RFC 6962 and should not be used for deep security analysis.[15]

VII. Discussion and Future Work

A. Specifications of Third-party Monitors

We suggest the following specifications for third-party monitors. First of all, a third-party monitor needs to disclose or provide an interface to receive the following information: *service delay*, *output limit*, and *log list*. We recommend that a monitor's service delay should be less than 4 days, which is the recommended lifetime of a short-term, automatically renewed (STAR) certificate [46]), so that (bogus) STAR certificates will be returned to users who only search unexpired ones.

Secondly, a third-party monitor should provide services through uniform APIs. This will reduce the complexity of developing a watcher, and also help domain owners to detect bogus certificates. A uniform API is suggested as below:

1) *Input and options:* a user may input the fingerprint of a certificate, or a domain with the options of expired certificates included or excluded, expected certificates or exactly-matching, and (pre)certificates deduplicated or not;

2) *Comparison:* a monitor compares a inquired domain with SAN:dNSNames in a case-insensitive way;

3) *Normal output for domain owners:* each (pre)certificate is returned, including CN, SAN:dNSNames, Issuer, SerialNumber, NotBefore, NotAfter, log-details, and cert-file, etc.;

4) *Opaque output for watchers:* Only a opaque set is returned, as *free* services for watchers; for example, each certificate is returned as only the digest of the quadruple (SerialNumber, Issuer, NotBefore, and NotAfter) and log-details. This optional

[15]Some bugs were discovered by the watcher in the preparation phase, so they were reported before January 25, 2020.

function enables public inspections by *light* watchers, while the monitors still charge domain owners for search services.

Finally, the access method to third-party monitors is designed similarly to log servers [30]. Client messages are sent to third-party monitors as HTTPS GET or POST requests. Afterward, the third-party monitor responds with a set of certificates (i.e., searched set) encoded as a JSON object. The third-party monitor should provide more secure and reliable TLS/HTTPS security services. For example, it (*i*) only accepts HTTPS-based access, (*ii*) only supports TLS 1.2 or above, (*iii*) uses CAA [21] and other mechanisms to limit which CAs can issue certificates for which websites to reduce MitM based on bogus certificates attack.

B. Defenses against Malicious Third-party Monitors

Analyzing the reasons for malicious behavior is infeasible and meaningless, so fault analysis is not considered for malicious monitors. While the light watcher is functionally capable of detecting any misbehavior of a monitor (either benign or malicious), the malicious action cannot be captured if the inquirer (e.g., the domain owner) does not run a watcher while the monitor is doing evil. Therefore, randomly running a watcher provides probabilistic protection against maliciously faulty monitors, acting as a deterrent, while running an ephemeral watcher (i.e., a light watcher with 1 input domain that runs for 1 period) whenever an inquiry occurs can detect malicious actions in a deterministic manner.

There are some monitors on the Internet that provide *public welfare* services (e.g., crt.sh and Facebook Monitor), while some provide *compensatory* services (e.g., Censys and SSLMate Spotter). The inquirer can run a watcher for free that includes only the requested monitor and monitors providing public welfare service, or pay to run one with more monitors participating, which provides greater capabilities. However, most inquirers would prefer the free service to the paid version. Therefore, we try to propose a free mode of operation for watchers. In this mode, the inquirer requests the search result (i.e., certificate set) from a monitor and inspects it by obtaining private sets from several other monitors for free, without revealing any data assets of them.

The *opaque output for watcher* mentioned in Section VII-A can be directly applied to light watchers to detect inconsistent results. The research on the applicability, deployment difficulty, scalability, and performance consumption of different private set structures/algorithms [25], [44], [18] to watchers is our future work. Building on the research results of this work, we generalize a private set manipulation watcher protocol based on Bloom filter (BF) [58], [17] to defend against malicious monitors, as follows:

Watcher protocol. First, the inquirer requests the certificates for domain d from a monitor, called *servicing monitor*. Second, the watcher inserts the certificates into a BF vector, denoted as BF_d^s. Third, the watcher sends the request to the selected *reference monitors* with domain d and the service limit (i.e., log list, service delay) of the servicing monitor. Fourth, each reference monitor m inserts the searched certificates within the service limit into a BF vector (denoted as BF_d^m) and returns it to the watcher. Fifth, the watcher constructs a reference BF vector (denoted as BF_d^r) by merging BF_d^m ($m \in \mathbb{M}$). It is worth noting that the voting method is also a potential solution to exclude irrelevant certificates from the private set. Finally, the watcher compares BF_d^s and BF_d^r bit by bit and alerts the inquirer if a bit is '1' for BF_d^r but '0' for BF_d^s, which means the servicing watcher missed some certificates.

Monitor workflow. Watchers increase the workload of the monitor by n times (n is the number of participating monitors). However, monitors can avoid searching large databases by storing and periodically updating the BF vectors of domains. Therefore, the extra work of the monitor is to insert the certificate into the BF vector after fetching it from log servers, in exchange for fast and easy responses to watchers.

Adaptation of subscription-based monitors. The BF-based watcher protocol has the additional advantage of being potentially useful for subscription services of monitors. By using this compressed data structure, it's possible to attach previous certificates (i.e., the BF_d^s of the last alert) to each monitor alert without displaying all certificates. The BF_d^s for a new alert is obtained by inserting the newly discovered certificates into the previous BF vector, based on which the watcher protocol can be completed.

VIII. Related Work

Security Analysis of CT. Several studies analyzed the security and/or privacy of CT. Stark et al. [51] studied the compliance, user experience, and potential risks of CT deployment. Li et al. [31], [32], [33] revealed the potential vulnerabilities of various CT components and exposed the unreliability of monitors in the wild. Oxford et al. [40] presented a methodology for formally evaluating quantitative aspects of the security of gossip protocols for CT. Kondracki et al. [26] proposed CTPOT, using a distributed honeypot system to analyze the CT bot ecosystem. Similarly, Pletinckx et al. [41] investigated the potential of CT logs as a data source for target reconnaissance using honeypot technology. Scheitle et al. [45] and Roberts et al. [42] discussed the leakage of domain information caused by CT. Meiklejohn et al. [34] summarized and categorized privacy concerns of SCT auditing in CT and existing solutions.

CT Enhancements. Some researchers aim to enhance CT. Kubilay et al. [27] introduced CertLedger, a blockchain-based improved CT-enabled PKI architecture, which aims to prevent split-view attacks and ensure optimal certificate/revocation transparency. Similarly, Wang et al. [57] deeply integrated blockchain and CT to reinforce the security guarantees of certificates. Hu et al. [23] introduced a novel transparency log system, called Merkle^2, which offers efficient monitoring and low-latency updates. Dirksen et al. [14] presented LogPicker to achieve mutual auditing amongst CT logs, thereby eliminating a trusted third party. Sun et al. [54] and Dahlberg et al. [10] respectively, proposed a lightweight monitoring scheme from the perspective of custom policy and verifiability. Sun et al. [55] merged the concepts of public-key encryption with keyword search (PEKS) and CT to ensure compatibility with transparency and privacy.

Certificate Parsing. Issued certificates may occasionally deviate from the format specification, including HTTPS certificates in the wild [15]. Kumar et al. [28] introduced ZLint, a linter that verifies certificates for compliance with technical standards and identifies errors in certificates issued by hundreds

of CAs. Chen et al. [7] introduced SBDT and discovered several bugs in certificate parsers by employing differential testing. Barenghi et al. [3] showed that 21.5% of the X.509 certificates are syntactically incorrect and proposed a more secure approach for parsing X.509 certificates. Debnath et al. [13] re-engineered the X.509 standard specification alleviating its design complexity, ambiguities, or under-specifications.

IX. Conclusion

In this paper, we present a scalable inspecting service, named CT watcher, to improve the reliability of CT. It can be run lightly by any interested party to detect the misbehavior of third-party monitors. Furthermore, the operator with sufficient resources can semi-automatically analyze potential failures leading to misbehavior through a full watcher. We measured 6 active third-party monitors via the full watcher. A total of 8.26M unique certificates were analyzed, and 14 design or implementation issues were detected, which resulted in 1.4M missing/irrelevant cases. It turns out that the current monitor still has some unnoticed exploitable faults, making it necessary to deploy watchers. Finally, we discussed specifications for services provided by third-party monitors to improve its cooperation with watchers further, as well as technical routes to resist malicious third-party monitors.

Acknowledgment

The authors thank Censys for providing the data support for this research. This work was partially supported by the National Key RD Plan of China under Grant 2017YFB0802100, the National Natural Science Foundation of China under Grant 62002011, the Youth Top Talent Support Program of Beihang University under Grant YWF-22-L-1272. In addition, Zeyan Liu and Fengjun Li were sponsored in part by NSF awards IIS-2014552, DGE-1565570, and the Ripple University Blockchain Research Initiative.

References

[1] Apple Inc., "Requirements for trusted certificates in iOS 13 and macOS 10.15," https://support.apple.com/en-hk/HT210176, 2019.

[2] ——, "Apple's Certificate Transparency Policy," https://support.apple.com/en-us/HT205280, 2021.

[3] A. Barenghi, N. Mainardi, and G. Pelosi, "Systematic Parsing of X.509: Eradicating Security Issues with a Parse Tree," *Journal of Computer Security*, vol. 26, no. 6, pp. 817–849, 2018.

[4] H. Birge-Lee, Y. Sun, A. Edmundson, J. Rexford, and P. Mittal, "Bamboozling Certificate Authorities with BGP," in *27th USENIX Security Symposium*, 2018, pp. 833–849.

[5] Certificate Transparency, "Working together to detect maliciously or mistakenly issued certificates," https://certificate.transparency.dev/, 2021.

[6] ——, "Our Story: Thank you to our amazing community," https://certificate.transparency.dev/community/#origins-grid, 2022.

[7] C. Chen, P. Ren, Z. Duan, C. Tian, X. Lu, and B. Yu, "SBDT: Search-Based Differential Testing of Certificate Parsers in SSL/TLS Implementations," in *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2023, pp. 967–979.

[8] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Academic press, 2013.

[9] crt.sh Inc, "Certificate Transparency log monitor of crt.sh," 2023, https://github.com/crtsh.

[10] R. Dahlberg and T. Pulls, "Verifiable Light-Weight Monitoring for Certificate Transparency Logs," in *Nordic Conference on Secure IT Systems*, 2018, pp. 171–183.

[11] T. Dai, H. Shulman, and M. Waidner, "Off-Path Attacks Against PKI," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 2213–2215.

[12] ——, "Let's Downgrade Let's Encrypt," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1421–1440.

[13] J. Debnath, S. Y. Chau, and O. Chowdhury, "On Re-engineering the X.509 PKI with Executable Specification for Better Implementation Guarantees," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1388–1404.

[14] A. Dirksen, D. Klein, R. Michael, T. Stehr, K. Rieck, and M. Johns, "LogPicker: Strengthening Certificate Transparency Against Covert Adversaries," *Proceedings on Privacy Enhancing Technologies*, vol. 4, pp. 1–19, 2021.

[15] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS Certificate Ecosystem," in *Proceedings of the 2013 conference on Internet measurement conference*, 2013, pp. 291–304.

[16] P. Eckersley, "A Syrian Man-In-The-Middle Attack against Facebook," 2011, https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook.

[17] R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, and J. Tillmanns, "Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters," in *20th Australasian Conference on Information Security and Privacy*, 2015, pp. 413–430.

[18] K. Frikken, "Privacy-Preserving Set Union," in *5th International Conference on Applied Cryptography and Network Security*, 2007, pp. 237–252.

[19] Google Inc., "Chrome Certificate Transparency Policy," https://googlechrome.github.io/CertificateTransparency/ct_policy.html, 2022.

[20] ——, "Feature: Support for commonName matching in Certificates (Removed)," https://chromestatus.com/feature/4981025180483584, 2022.

[21] P. M. Hallam-Baker and R. Stradling, "IETF RFC 6844 - DNS Certification Authority Authorization (CAA) Resource Record," 2013.

[22] Heather Adkins, "An update on attempted man-in-the-middle attacks," 2011, https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html.

[23] Y. Hu, K. Hooshmand, H. Kalidhindi, S. J. Yang, and R. A. Popa, "Merkle^2: A low-latency transparency log system," in *IEEE Symposium on Security and Privacy*, 2021, pp. 285–303.

[24] S. Kent, "IETF Draft - Attack and Threat Model for Certificate Transparency," 2018.

[25] L. Kissner and D. Song, "Privacy-Preserving Set Operations," in *Annual International Cryptology Conference*, 2005, pp. 241–257.

[26] B. Kondracki, J. So, and N. Nikiforakis, "Uninvited Guests: Analyzing the Identity and Behavior of Certificate Transparency Bots," in *31st USENIX Security Symposium*, 2022, pp. 53–70.

[27] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar, "CertLedger: A New PKI Model with Certificate Transparency Based on Blockchain," *Computers & Security*, vol. 85, pp. 333–352, 2019.

[28] D. Kumar, Z. Wang, M. Hyder, J. Dickinson, G. Beck, D. Adrian, J. Mason, Z. Durumeric, J. A. Halderman, and M. Bailey, "Tracking Certificate Misissuance in the Wild," in *IEEE Symposium on Security and Privacy*, 2018, pp. 785–798.

[29] B. Laurie, A. Langley, and E. Kasper, "RFC 6962: Certificate Transparency," 2013.

[30] B. Laurie, E. Messeri, and R. Stradling, "RFC 9162: Certificate Transparency Version 2.0," 2021.

[31] B. Li, D. Chu, J. Lin, Q. Cai, C. Wang, and L. Meng, "The Weakest Link of Certificate Transparency: Exploring the TLS/HTTPS Configurations of Third-Party Monitors," in *18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications*, 2019.

[32] B. Li, J. Lin, F. Li, Q. Wang, Q. Li, J. Jing, and C. Wang, "Certificate Transparency in the Wild: Exploring the Reliability of Monitors," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[33] B. Li, J. Lin, F. Li, Q. Wang, W. Wang, Q. Li, G. Cheng, J. Jing, and C. Wang, "The Invisible Side of Certificate Transparency: Exploring the Reliability of Monitors in the Wild," *IEEE/ACM Transactions on Networking*, vol. 30, no. 2, pp. 749–765, 2021.

[34] S. Meiklejohn, J. DeBlasio, D. O'Brien, C. Thompson, K. Yeo, and E. Stark, "SoK: SCT Auditing in Certificate Transparency," in *Proceedings on Privacy Enhancing Technologies*, 2022, pp. 336–353.

[35] B. Morton, "More Google Fraudulent Certificates," 2014, https://www.entrust.com/google-fraudulent-certificates/.

[36] Mozilla Foundation, "Public Suffix List," 2022, https://publicsuffix.org/list/.

[37] L. Nordberg, D. K. Gillmor, and T. Ritter, "Gossiping in CT," https://datatracker.ietf.org/doc/html/draft-ietf-trans-gossip-05, 2018.

[38] D. O'Brien, "Chrome CT 2021 Plans," https://groups.google.com/a/chromium.org/g/ct-policy/c/4puGir9pNFA/m/1caF3ilrBQAJ, 2021.

[39] Opsmate Inc., "Certificate Transparency Log Growth," https://sslmate.com/labs/ct_growth/, 2022.

[40] M. Oxford, D. Parker, and M. Ryan, "Quantitative Verification of Certificate Transparency Gossip Protocols," in *IEEE Conference on Communications and Network Security*, 2020, pp. 1–9.

[41] S. Pletinckx, T.-D. Nguyen, T. Fiebig, C. Kruegel, and G. Vigna, "Certifiably Vulnerable: Using Certificate Transparency Logs for Target Reconnaissance," in *IEEE European Symposium on Security and Privacy Workshops*, 2023.

[42] R. Roberts and D. Levin, "When Certificate Transparency is Too Transparent: Analyzing Information Leakage in HTTPS Domain Names," in *ACM Workshop on Privacy in the Electronic Society*, 2019, pp. 87–92.

[43] Salvatore S. Mangiafico, "Goodness-of-Fit Tests for Nominal Variables," 2016, https://rcompanion.org/handbook/H_03.html.

[44] Y. Sang and H. Shen, "Efficient and Secure Protocols for Privacy-Preserving Set Operations," *ACM Transactions on Information and System Security*, vol. 13, no. 1, pp. 1–35, 2009.

[45] Q. Scheitle, O. Gasser, T. Nolte, J. Amann, L. Brent, G. Carle, R. Holz, T. C. Schmidt, and M. Wählisch, "The Rise of Certificate Transparency and its Implications on the Internet Ecosystem," in *Proceedings of the Internet Measurement Conference*, 2018.

[46] Y. Sheffer, D. Lopez, O. G. de Dios, A. Pastor, and T. Fossati, "RFC 8739: Support for Short-Term, Automatically Renewed (STAR) Certificates in the Automated Certificate Management Environment (ACME)," 2020.

[47] SSLMate Inc., "How Cert Spotter Parses 255 Million Certificates," https://sslmate.com/blog/post/how_certspotter_parses_255_million_certificates, 2017.

[48] ——, "Security incident report," 2018, https://sslmate.com/resources/certificate_authority_failures.

[49] ——, "Certificate Transparency log monitor of SSLMate," 2020, https://github.com/SSLMate/certspotter.

[50] ——, "One Billion Certificates, At Your Fingertips," https://sslmate.com/ct_search_api/, 2022.

[51] E. Stark, R. Sleevi, R. Muminovic, D. O'Brien, E. Messeri, A. P. Felt, B. McMillion, and P. Tabriz, "Does Certificate Transparency Break the Web? Measuring Adoption and Error Rate," in *IEEE Symposium on Security and Privacy*, 2019.

[52] E. Stark and C. Thompson, "Opt-in SCT auditing," https://docs.google.com/document/d/1G1Jy8LJgSqJ-B673GnTYIG4b7XRw2ZLtvvSlrqFcl4A, 2020.

[53] R. Stradling and E. Messeri, "Certificate Transparency: Domain Label Redaction," 2017.

[54] A. Sun, B. Li, H. Wan, and Q. Wang, "PoliCT: Flexible Policy in Certificate Transparency Enabling Lightweight Self-monitor," in *Applied Cryptography and Network Security Workshops*, 2021, pp. 358–377.

[55] A. Sun, B. Li, Q. Wang, H. Wan, J. Lin, and W. Wang, "Semi-CT: Certificates Transparent to Identity Owners but Opaque to Snoopers," in *IEEE Symposium on Computers and Communications*, 2023, pp. 1207–1213.

[56] K. S. University, "SPSS Tutorial: Chi-Square Test of Independence," 2019, https://libguides.library.kent.edu/spss/chisquare.

[57] Z. Wang, J. Lin, Q. Cai, Q. Wang, D. Zha, and J. Jing, "Blockchain-based Certificate Transparency and Revocation Transparency," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[58] Wikipedia, "Bloom filter," https://en.wikipedia.org/wiki/Bloom_filter, 2023.

[59] K. Wilson, "Distrusting New CNNIC Certificates," 2015, https://blog.mozilla.org/security/2015/04/02/distrusting-new-cnnic-certificates/.

APPENDIX A

This appendix summarizes the exploitation of the bugs discovered in this work (see Section VI-C for details).

Output limit. The APIs of Entrust Search, crt.sh, and Facebook Monitor have output limits, which means only the oldest n certificates for the inquired domain are returned. For a domain with more than n certificates, newly issued certificates will not be returned. Similarly, Google Monitor returns certificates with a matching CN or SAN:dNSName in lexicographical order compared from the top-level domain to the subdomains, but it tends to miss certificates binding a lexicographically greater domain. So an attacker could target such domains with a great number of legitimate certificates, and then bogus certificates would be (probably) invisible from these monitors.

Unmonitored or unprocessed log. A monitor pre-selects a set of logs to collect data, so not all public logs are monitored (e.g., Censys, and Entrust Search). Therefore, an attacker can evade detection by submitting the bogus certificate to unmonitored logs. Furthermore, even when a log is monitored, if the appended records exceed the monitor's processing capacity, it has to create a backlog to store *unprocessed* certificates (as shown in Fig. 6 of Appendix E). For example, Censys maintains a large backlog for the "Google Argon 2020" log. Thus, an attacker might submit a bogus certificate to some carefully-selected logs, which are not monitored by some monitors or have brought backlogs, and then this certificate will be accepted by CT-compliant browsers but invisible (at least for a long period).

Defective certificate look-up. The missing certificates of some monitors exhibit distinctive patterns of certificate contents. For example, Entrust Search and Censys compare an input domain and the parsed SAN:dNSName(s) in a *case-sensitive* way, while Censys does not return certificates with a large number of SAN:dNSNames (e.g., a certificate issued to CDN servers with SAN:dNSNames containing more than 9,000 characters in total). An attacker could encode the target domain as a SAN:dNSName by mixed upper and lower-case letters (e.g., WwW.fAceBOoK.cOm) or accompanied with many nonexistent domains, to conceal a bogus certificate.

APPENDIX B

The implementation of the scraper to the participating third-party monitors is as follows:

Censys. The scraper uses the BigQuery interface and prepares SQL statements to obtain consistent search results returned by the web interface. The SQL statement used is equivalent to the "parsed.subject.common_name:domain OR parsed.names:domain OR parsed._expanded_names:domain OR parsed.extensions.subject_alt_name.dns_names:domain" in the web service. It uses the "unexpired" and "ct" tags to exclude expired certificates and certificates obtained through active scanning. We also filter out the certificates recorded only in test logs (i.e., "Testtube Log" and "Comodo Dodo").

TABLE VIII: The actual merge delay for the 24 unretired logs that comply with Chrome's CT policy.

Log Server	Merge Delay Range	Average Merge Delay
Google Argon (2022)	1s - 137s	4.17s
Google Argon (2023)	1s - 135s	2.86s
Google Argon (2024)	1s - 125s	2.95s
Google Xenon (2022)	1s - 247s	4.26s
Google Xenon (2024)	1s - 125s	3.67s
DigiCert Nessie (2022)	3547s - 3604s	3585s
DigiCert Nessie (2023)	590s - 605s	602s
DigiCert Nessie (2024)	546s - 603s	598s
DigiCert Nessie (2025)	600s - 601s	600s
DigiCert Yeti (2023)	600s - 601s	600s
DigiCert Yeti (2024)	600s - 602s	601s
DigiCert Yeti (2025)	600s - 601s	600s
Let's Encrypt Oak (2022)	3s - 138s	14.7s
Let's Encrypt Oak (2023)	1s - 131s	12.8s
Let's Encrypt Oak (2024h1)	1s - 127s	3.25s
Let's Encrypt Oak (2024h2)	7200s	7200s
Cloudflare Nimbus (2022)	480s - 1920s	938s
Cloudflare Nimbus (2023)	120s - 1440s	992s
Cloudflare Nimbus (2024)	480s - 2520s	1035s
TrustAsia Log (2022)	10s - 731s	419s
TrustAsia Log (2023)	9s - 659s	76.8s
TrustAsia Log (2024)	1660s - 3601s	2400s
Sectigo Mammoth	92s - 600s	312s
Sectigo Sabre	13s - 600s	548s

crt.sh. The scraper uses the PostgreSQL database interface and the same SQL statement to obtain the same results as querying the web service. It excludes the certificates matched by O and OU, the scanned certificates whose *log-details* is null, and the certificates only recorded in the "Comodo Dodo" log.

Entrust Search. The scraper crawls web pages directly.

Facebook Monitor. The scraper uses the graph API to obtain the certificate in paging (the page size is set to 5,000). It also excludes expired certificates.

Google Monitor. The scraper crawls certificates from web pages. It first collects certificate ID (Base64 encoding of SHA256 fingerprint) from the overview page and then visits the certificate page for detailed information. It also excludes expired certificates.

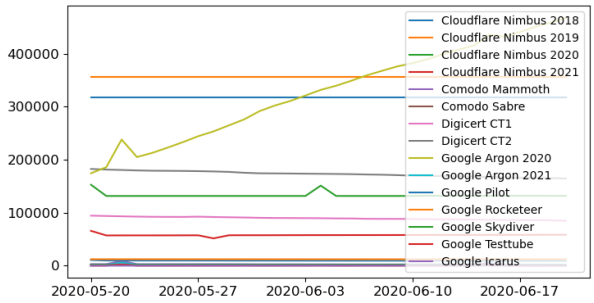
SSLMate Spotter. The scraper uses the JSON API to fetch the certificates, which does not return SerialNumber. So, we parse the certificate to extract the information.

APPENDIX C

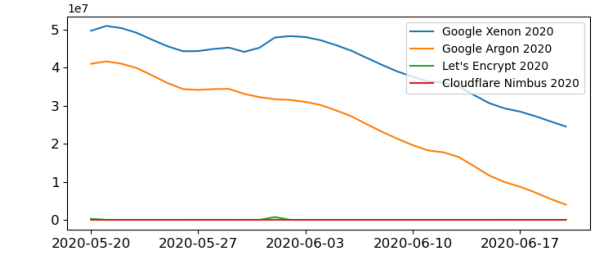
Table IX provides some examples of labels available to be attached to certificates when implementing the semi-automated fault analyzer, most of which are included in our prototype. TABLE X shows the manually constructed trigger features guided by the high-ranked features output by the random forest.

APPENDIX D

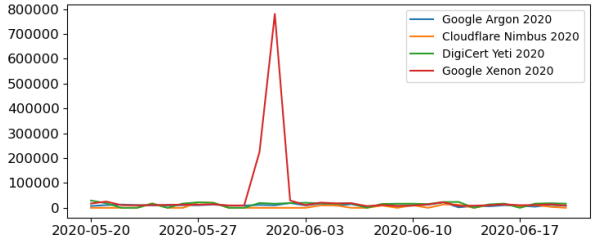
This appendix presents the actual merge delay measured on December 29, 2022, for the 24 unretired logs that are compliant with Chrome's CT policy, as shown in TABLE VIII. We queried the latest STH of the log about every second for about 2 hours and counted how many different STHs were obtained during the period. For some log servers with actual merge delays of up to tens of minutes, a longer trace was performed, this time once every minute for 10 hours. The results show that the actual average merging delay of 8 logs is less than 1 minute, the delay of 13 logs is within 20 minutes, the delay of 2 logs is within 1 hour, and the maximum delay is 2 hours.



(a) Censys certificate backlog.



(b) Crt.sh certificate backlog.



(c) SSLMate Spotter certificate backlog.

Fig. 6: The backlog of certificates in the monitors that provide its log list.

APPENDIX E

This appendix contains some details of the experimental evaluation and analysis. TABLE XI presents the search results of 4 super domains, and TABLE XII presents information about 7 domains used to analyze Google Monitor's output limits. Fig. 6 illustrates the backlogs of Censys, crt.sh and SSLMate Spotter.

TABLE IX: The example of labels attached to certificates in the implementation of the semi-automated fault analyzer.

Classification	Label
Certificate	Issuer NotBefore NotAfter ExtendedKeyUsage (e.g., client authentication and email protection) lifetime (i.e., $NotAfter - NotBefore$) number of SAN:dNSNames the average length of its SAN:dNSName whether it is a redacted certificate (True/False)
Submission	first submitted time time difference from first submitted to last submitted time difference from NotBefore to first submitted number of SCTs number of operators issuing these SCTs average size of logs submitted average daily growth of logs submitted logged format
Domain inquired	eTLD of the domain inquired number of target domains minimum level of target domains whether target domains are mixed case (True/False) whether target domains are IDNs (True/False) whether target domains contain the character '-'/'_' (True/False) average size of reference sets the difference in size between the reference sets the number of segments (i.e., pages) the monitor returns the result

TABLE X: Manually constructed trigger features based on the guidance of high-ranked features.

High-ranking Feature		Trigger Feature
Censys-1: "number of SAN:dNSNames"	→	F4: Its SAN:dNSNames contain over 8,830 characters.
Entrust-5: "average daily growth of logs submitted"	→	F2: It only submitted to the busiest logs.
Facebook-1: "average size of reference sets"	→	F9: The search results of the *domain inquired* need to be returned on multiple segments.
Google-3: "lifetime"	→	F11: Its lifetime is less than 91 days.

TABLE XI: The search results of 4 super domains.

	Censys	crt.sh	Entrust Search	Facebook	Google	SSLMate Spotter
amazonaws.com	1,178,292	⊘	⊘	31,046	374,693	⊘
zendesk.com	2,040,024	⊘	⊘	13,311	315,958	⊘
azure.com	3,570,640	⊘	⊘	12,516	265,965	⊘
netflix.com	51,641	52,193	⊘	24,214	45,753	⊘

TABLE XII: The statistics of 7 domains for which Google Monitor only returns partial results.

	Number of Google missing certificates	Number of relevant certificates	Number of short-lived certificates	Percentage
azure.com	3,160,068	4,808,933	-	-
zendesk.com	1,587,280	4,361,666	-	-
amazonaws.com	801,983	3,425,695	-	-
netflix.com	6,278	522,177	521,287	99.83%
wixsite.com	16,362	36,188	36,161	99.99%
sheridanc.on.ca	3,384	82,600	82,516	99.90%
ugm.ac.id	131	44,588	43,148	96.77%

APPENDIX F
ARTIFACT APPENDIX

A. Description & Requirements

The submitted artifact is the light/full watcher described in this paper (link: https://github.com/PKIexr/CT-watcher or https://doi.org/10.5281/zenodo.10148256). It includes the inconsistency analyzer and the machine-learning model in the fault analyzer. We also provide the data obtained during the experiments (January 25, 2020 - March 16, 2021) to demonstrate the functions.

Launching this artifact requires Python3, as well as psycopg2-binary, pyOpenSSL, urllib3, publicsuffixlist, censys, schedule, numpy, pypi-json, scikit-learn, pandas, glob2 and matplotlib. The accounts of inspected third-party monitors and the corresponding access tokens are also required.

B. Experiment Workflow

1) Light watcher: Light watcher is used to detect misbehaviour of third-party monitors. It takes a list of domains as input and outputs the inconsistency between the search results of each third-party monitor and the complete certificate set. Algorithm 1 provides the pseudocode of the light watcher (i.e., inconsistency analyzer), which is described in Sections V-B and V-C. Perform the functions of a light watcher by running "Inconsistency_Analyzer".[16]

Before running the light watcher, you need to configure the domain input in the "domains.csv" file and the necessary parameters in the "config.py" file as below: the data storage directory ("DATA_ROOT_FOLDER"), the number of tracking periods ("PERIOD_NUM"), monitors inspected ("MONITOR_INVOLVED), the service information ("MONITOR_CONFIG") and access tokens ("FACEBOOK_TOKEN" and "SSLMATE_TOKEN").

2) Full Watcher: Full watcher is used to detect and also analyze the misbehaviour of third-party monitors. Its input is a list of domains, while its output is a set of inconsistent certificates and trigger features. To effectively demonstrate the functions of the full watcher, we provide the 52-day tracking data for 4,000 domains (as described in Section VI). The fault analyzer analyzes the provided data and outputs several ranked feature lists. TABLE X shows examples of high-ranked features, which are further analyzed in Section VI-C. Run "Fault_Analyzer" to extract and rank features on the provided data (or any data you obtain).

C. Major Claims

- (C1): Light watchers detect misbehaviour of third-party monitors (i.e., returning incorrect search results for some certain domain inputs). Experiment E1 illustrates this claim.
- (C2): The machine-learning model extracts features, ranks features, and guides the manual analysis to find trigger features. Experiment E2 illustrates this claim.

[16]In the prototype implementation, the termination condition is determined by the number of tracking periods set by the operator.

D. Evaluation

1) Experiment (E1): **[Light watcher[17]]** inspects the search services of three third-party monitors (i.e., crt.sh, Facebook Monitor and SSLMate Spotter). On receiving the domains of interest, the light watcher will output the number of certificates that are either missing or returned incorrectly by the third-party monitors for each domain.

[Preparation-1: Account for third-party monitors] Fill in the API key of SSLMate Spotter ("SSLMATE_TOKEN") and the APP token of Facebook Graph API ("FACEBOOK_TOKEN") in "config.py". For SSLMate Spotter, you need to obtain the API key from https://sslmate.com/certspotter/. For Facebook Graph API, you need to obtain the APP token from their developer tools at https://developers.facebook.com/tools/explorer/. For crt.sh, there is no need to register an account.

[Preparation-2: Parameter setting] The parameters are also configured in "config.py". Set "DATA_ROOT_FOLDER" as the directory (e.g., "data/") where the data files are stored. Set the number of periods that the light watcher needs to execute as "PERIOD_NUM". In addition, we have set the "MONITOR_INVOLVED" and "MONITOR_CONFIG" by default. Operators can modify these parameters as needed.

[Preparation-3: Domain input] Please fill in the domains in the "domains.csv" file as the input to the light watcher.

[Execution] Run "python3 main.py".

[Results] Light watcher first outputs "Collecting certificates!!!" and the estimated time to start collecting certificates. It then outputs the domain being inquired, which takes a long time. After that, it outputs "Construct the reference set!!!", "Construct the irrelevant set and the missing set!!!" and "Classify missing certificates!!!", indicating the steps it is executing. Finally, it outputs "Inspection Result:" along with the size of the reference set, searched set and missing set for each trigger domain, third-party monitor and period.

The raw search data are stored in the folder named "*./period/monitor*/RawData/", while the searched sets are in "*./period/monitor*/ProcessedCert/". The irrelevant sets and the missing sets are in "*./period/monitor*/IrrelevantCert/" and "*./period/monitor*/MissingCert/". Additionally, any missing certificates resulting from a monitor's output limitations are stored in "*./period/monitor*/ServiceLimit/".

2) Experiment (E2): **[Full Watcher[18]]** extracts and ranks features of missing certificates (or irrelevant certificates) of third-party monitors to guide the manual analysis.

[Preparation: Dataset] Machine-learning models require extensive data input. To demonstrate these functions, we provide the data obtained during our experiment (January 25, 2020 - March 16, 2021) at link: https://drive.google.com/file/d/1ivW2GKU47JjKwxG06SJgszdqUYWJ9T9b/view. Download, unzip and place it in the directory "Watcher/".

[Execution & Results] (*1*) For from-scratch feature extraction, run "python json2csv.py". (*2*) The pre-saved features are saved in "./CSV/" (*3*) If using pre-saved features, run "python

[17]"Watcher/Inconsistency_Analyzer/"

[18]"Watcher/Fault_Analyzer/"

supervised.py i 1", and "i" for the monitor id (0-5). It will show a preview of the data distribution and example data columns at first. Then it will run several machine learning classifiers. (*4*) For attributing the feature importance, run "python reasoning.py". The importance of the missed and delayed certificates will be recorded in "reasons.csv".

E. Notes

Google Monitor is currently out of service, and Entrust Search enforces an anti-crawler mechanism. Besides, obtaining the required data from Censys is somehow expensive. Therefore, this watcher prototype contains only 3 active third-party monitors, namely crt.sh, Facebook Monitor and SSLMate Spotter.

In the experiments of this paper, in order to obtain large amounts of data, we used a dedicated API for researchers to access Censys, with multiple (paid) accounts and multiple threads. Thus, it is extremely difficult (or even impossible) to obtain such many certificates (i.e., nearly 1 million certificates from 4,000 domains) as shown in Section VI by using the open-source prototype of light watcher, and the experimental results cannot be strictly mapped. However, the open-source light watcher still plays an important role in detecting potential failures of third-party monitors, as described in Section IV-C.

Algorithm 1: Inconsistency analyzer

input : $domains$, $service\ information$;
output: inspection results of monitors for each day;
N = 1;
while *watcher not terminated* **do**
  for $d \in domains$ **do**
    for $m \in \mathbb{M}$ **do**
      Obtain $\mathbb{S}^i_{d,m}$ from m;
      Insert $\mathbb{S}^i_{d,m}$ into $\mathbb{T}_d$;
      for $c \in \mathbb{T}_d$ **do**
        Update the $log\text{-}details$, $SubmittedTime$, $vote$, etc. for c;
      end
    end
    for $i \in [1, N]$ **do**
      $\tilde{\mathbb{R}}^i_d = \mathbb{T}_d - \mathbb{U}^i_d - \mathbb{E}^i_d$;
      for $c \in \tilde{\mathbb{R}}^i_d$ **do**
        if c *is not irrelevant* **then**
          Insert c into $\mathbb{R}^i_d$;
        end
      end
      for $m \in \mathbb{M}$ **do**
        $\mathbb{S}^{+\ i}_{d,m} = \mathbb{S}^i_{d,m} - \mathbb{R}^i_d$;
        $\mathbb{S}^{-\ i}_{d,m} = \mathbb{R}^i_d - \mathbb{S}^i_{d,m}$;
        for $c \in \mathbb{S}^{-\ i}_{d,m}$ **do**
          if c *is missed due to service limitations* **then**
            Insert c into $\hat{\mathbb{S}}^{-\ i}_{d,m}$;
          end
        end
        if $\mathbb{S}^{+\ i}_{d,m}$ *or* $\mathbb{S}^{-\ i}_{d,m}$ *is not empty* **then**
          $Result[monitor][i][d] \leftarrow [SearchedTime_i, \mathbb{S}^i_{d,m}, \mathbb{S}^{+\ i}_{d,m}, \mathbb{S}^{-\ i}_{d,m}, \hat{\mathbb{S}}^{-\ i}_{d,m}]$;
        end
      end
    end
  end
  Output $Result$;
  Wait until the next period;
  N++;
end