Where the Wild Warnings Are: Root Causes of Chrome HTTPS Certificate Errors

Mustafa Emre Acer
Google Inc.

Emily Stark
Google Inc.

Adrienne Porter Felt
Google Inc.

Sascha Fahl
Leibniz University Hannover

Radhika Bhargava
Purdue University

Bhanu Dev
International Institute of Information Technology Hyderabad

Matt Braithwaite
Google Inc.

Ryan Sleevi
Google Inc.

Parisa Tabriz
Google Inc.

ABSTRACT

HTTPS error warnings are supposed to alert browser users to network attacks. Unfortunately, a wide range of non-attack circumstances trigger hundreds of millions of spurious browser warnings per month. Spurious warnings frustrate users, hinder the widespread adoption of HTTPS, and undermine trust in browser warnings. We investigate the root causes of HTTPS error warnings in the field, with the goal of resolving benign errors.

We study a sample of over 300 million errors that Google Chrome users encountered in the course of normal browsing. After manually reviewing more than 2,000 error reports, we developed automated rules to classify the top causes of HTTPS error warnings. We are able to automatically diagnose the root causes of two-thirds of error reports. To our surprise, we find that more than half of errors are caused by client-side or network issues instead of server misconfigurations. Based on these findings, we implemented more actionable warnings and other browser changes to address client-side error causes. We further propose solutions for other classes of root causes.

1 INTRODUCTION

HTTPS certificate error warnings are supposed to protect users by alerting them to network attacks. Instead, users see hundreds of millions of warnings per month in the absence of real attacks. A user might see warnings when she connects her phone to her office's network, or when a server administrator forgets to update a certificate. Spurious HTTPS warnings are problematic because:

- Errors are a poor user experience. They frighten people and prevent them from completing important tasks.
- Warnings hinder adoption of HTTPS. Developers are frustrated when they switch a website to HTTPS and then hear complaints from customers about errors.
- Over time, people pay less attention to warnings if they believe them to be false alarms [28, 30]. We want people to pay attention for the occasion when it really is an attack.

Despite these problems, we can't simply get rid of HTTPS error warnings altogether. HTTPS certificate warnings are foundational to the security of the web. When an attacker intercepts a connection, the browser detects the attacker's invalid certificate chain and warns the user about it. If the user clicks through the warning, then the attacker can read and tamper with data on the website.

Browser vendors have tried to improve HTTPS error warnings with changes to UI and storage policies [15, 16, 29]. Although their improvements have increased warning adherence, their work is not done: HTTPS warnings still suffer from the core problem of false alarms [27]. Yet, the warnings must remain to protect users under attack. Our goal is to balance both needs: remove spurious warnings without impinging on legitimate warnings.

To address spurious warnings, we must first understand their root causes. We investigate the most common causes of certificate errors in a large-scale dataset of HTTPS error warnings encountered in the field. Chrome users volunteered to share HTTPS error reports with us, averaging one million reports per day over one year. We developed an analysis pipeline that automatically classifies errors by their root cause when possible. Two-thirds of reports can be automatically classified in this way, and we characterize the remainder by manually reviewing a sample.

We find that more than half of certificate errors are due to non-attack network interception or problems with the client. Previous work focused only on the role of server misconfigurations [9, 12, 14, 18], but we show that client and network health are equally as important. Further, two types of errors — insufficient intermediates and incorrect client clocks — are the biggest individual error causes for Android and Windows clients. These two error classes are good targets for mitigations that prevent unnecessary warnings. We also find that government websites are disproportionately responsible for server errors: 65% of the most-visited websites with warnings are run by governments. The prevalence of errors on government websites is alarming because it trains users to click through warnings on important sites such as tax payment portals.

Many of these problems can be mitigated by building more actionable warnings in the browser or investing in other client-side engineering solutions. We propose several mitigations and implement four of them for Google Chrome, an open source browser

CCS'17, , Oct. 30–Nov. 3, 2017, Dallas, TX, USA.

ACM ISBN 978-1-4503-4946-8/17/10.
https://doi.org/10.1145/3133956.3134007

with over two billion active users. Altogether, we expect these four mitigations to replace about 25% of error warnings.

Contributions. Our primary contributions are:

- We collect and analyze a large-scale, longitudinal dataset of certificate warnings encountered in the wild. We perform an in-depth study of each class of error.
- To our knowledge, we are the first to identify and quantify the significant role that client and network misconfigurations play in HTTPS error warnings.
- Among misconfigured sites, we characterize the "worst offenders": the sites that cause the most certificate warnings in Chrome. This group is dominated by government websites.
- To our knowledge, we are the first to deploy technical measures to reduce the frequency of spurious HTTPS error warnings. Additionally, we propose and discuss further mitigations for future work.

2 BACKGROUND

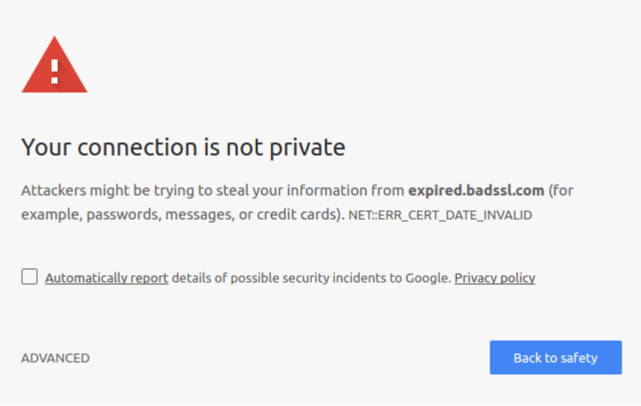
HTTPS protects the integrity and confidentiality of web traffic in transit, even in the presence of an active network attacker. Historical network attackers include governments, ISPs, roommates, criminals on public networks, and others.

When a browser sets up an HTTPS connection with a server, the browser must check that it's communicating with the actual server and not a network attacker. (Without this identity check, a network attacker could pretend to be the server to capture the decryption key.) At a high level, this process has three parts:

(1) At some point in the past, the server administrator obtained a certificate signed by a Certificate Authority (CA).
(2) After setting up a TCP connection, the server provides the signed certificate to the browser.
(3) The browser attempts to build a chain of trust from the certificate to a root certificate on the client. The *root trust store* on the client contains a set of root certificates from trusted CAs. The browser also performs other checks, e.g. to make sure that the certificate has the appropriate hostname in it and that the certificate is not yet expired. Any failure means that the browser is unsure of the identity of the server. This step is called *certificate validation.*

If everything goes well, the HTTPS page loads. If a certificate validation check fails, the browser shows an HTTPS error warning (Figure 1) without loading any of the page content.

TLS proxies slightly complicate this story. A wide range of middleboxes (e.g., corporate network firewalls and school content filters) and software (e.g., anti-virus software and debugging tools) want to intercept HTTPS traffic for various legitimate purposes. This is accomplished by installing a root certificate from the TLS proxy vendor into the client's trust store. The proxy will then issue new certificates for all of the client's incoming web traffic, signed by the proxy's root certificate. This is a widespread but contentious practice because TLS proxies can introduce vulnerabilities [10].



Your connection is not private

Attackers might be trying to steal your information from **expired.badssl.com** (for example, passwords, messages, or credit cards). NET::ERR_CERT_DATE_INVALID

□ Automatically report details of possible security incidents to Google. Privacy policy

Figure 1: An HTTPS warning in Chrome 58. The checkbox controls whether a certificate error report is uploaded when the user encounters such a warning.

3 DATASET

Millions of Google Chrome users volunteer to upload error reports when they encounter HTTPS or Safe Browsing warnings. Our study is based on a large-scale sample of these reports.

3.1 Our volunteers

Users volunteer to share certificate error reports by checking a checkbox on the HTTPS warning page (Figure 1). Once checked, the setting is remembered in a preference associated with the user's Chrome profile. The user can stop participating in the program from Chrome settings, or by un-checking the checkbox on any subsequent warning page. The same setting can be toggled by a similar checkbox on Safe Browsing warnings in Chrome [8].

The size of our dataset (a million reports a day) suggests that it represents a large swath of browser users. For privacy reasons, however, we intentionally do not analyze or retain identifying information to try to characterize our volunteers (discussed further in Section 3.4).

3.2 Sample dates and size

We enabled the reporting service in Google Chrome 44, which was released to Chrome's stable channel in July 2015. We focus most closely on the data from April 2016 through March 2017, which we refer to as the *Annual Reports sample*, containing 361,198,513 reports.[1]

All Google Chrome release channels (canary, dev, beta, and stable) [2] upload reports. Most users are on the stable channel, which enjoys wide deployment. To avoid overwhelming our report processing pipeline, stable Chrome sends a given certificate report to the server with only 20% probability. For analysis purposes, we restrict our discussion to reports from Chrome's stable channel, which we believe to be the most representative.

[1] Unfortunately, we cannot release the percentage of total Chrome browsing activity that certificate warnings represent. In prior work, Akwahe et al. found a false warning rate of 1.54% [9].

3.3 Report contents

Each Chrome certificate error report contains:

- The hostname that the user was trying to access
- The certificate chain that Chrome received from the server
- The certificate chain that the client built while attempting to validate the certificate
- The user's local system time at the time the error occurred
- The certificate validation error(s) that Chrome encountered (for example, the certificate was expired or did not match the requested hostname)
- Whether Chrome customized the warning page for the particular type of error or showed a generic HTTPS warning
- Whether the user clicked through the warning to continue to the site
- The browser's User-Agent string
- Relevant Chrome field trials, which are Chrome features enabled on an experimental basis [22]

Notably, each report contains *two* certificate chains: the chain presented by the server, and the chain built by the client. They can differ for a variety of reasons. One common example is that a server presented a chain to a root certificate that a client doesn't trust, and the client tried to build an alternate path to a root that it does trust.

3.4 Privacy

Certificate error reports may contain private information. For example, a certificate from an intranet might include the name of the company's system administrator or a testing computer's hostname. We take several steps to protect and respect our volunteers' privacy.

3.4.1 No identifiers. The reports are *not* associated or stored with any user or client identifiers. If someone uploads multiple reports, we do not associate the reports with the same user.

3.4.2 Anonymous data retention. We strip all potentially private data from the reports after they are two weeks old. We remove all certificate chains that have not been seen publicly by Googlebot, retaining only a SHA256 hash of the chain. We retain publicly resolvable hostnames and generic identifiers like "localhost"; otherwise, we replace hostnames with coarse alternatives like "Intranet host" and "Intranet IP." We tokenize User-Agent strings and retain only the major version number (e.g., "58" rather than "58.0.3029.96"), the locale, the operating system, and the platform.

3.4.3 Protection in transit. The reports are protected in transit on the network. Because these reports are sent to help investigate conditions that prevent the user from sending HTTPS requests, the reports themselves cannot be reliably sent over HTTPS. For example, if the user's local system clock is set incorrectly, it may prevent a report about the condition from being sent to Google over HTTPS because the connection to Google may appear to be using an expired certificate. Therefore, Chrome sends reports to an HTTP URL. The report payload is encrypted with a public key that ships with Chrome, to prevent a network attacker from eavesdropping on private information that may appear in the reports.

3.5 Limitations

Field data has inherent limitations, some of which we are able to mitigate (and some which we cannot). Still, we feel that the large scale of our dataset and its *in situ* collection method yield results with strong ecological and external validity.

3.5.1 Active attacks. An active network attacker could block reports from being uploaded. The absence of active attacks in our dataset does not mean that active attacks do not occur. For this reason, we investigate unintentional misconfigurations, rather than attempting to uncover active attacks.

3.5.2 Upload failures. In addition to active attacks, various network conditions can prevent reports from being uploaded. If the upload fails, Chrome does not persist or retry reports.[2]

3.5.3 Channel identification. We want to restrict our analysis to the stable channel, which has the most representative user population. Unfortunately, we stripped the full version strings from historical data (Section 3.4). We therefore use a heuristic to identify reports from the stable channel: for a given date, we only consider reports from that date's stable release or older. (For example, the stable release on October 1, 2016 was Chrome 53, so for that date we analyze reports from Chrome 53 or older.) We believe that this will filter out nearly all reports from non-stable channels, but a few non-stable reports might remain.

3.5.4 Chromium forks. Our dataset also contains a small number of reports from other browsers based on the Chromium source code[3]. Because we removed User-Agent strings, it is not possible to remove these from the historical dataset. We throttled Chrome stable reports to 20%, but other Chromium browsers did not. As a result, they are over-represented in the dataset relative to reports from official stable Chrome. By inspecting User-Agent strings from May 2017 reports that have not yet been anonymized, we believe that less than 2% of reports come from other Chromium browsers.

3.5.5 Volunteer bias. We receive reports only from people who choose to participate in the program. It is possible that they are not representative of all Chrome users. However, the considerable size of our dataset suggests that it represents a large swath of browser users.

3.6 Telemetry data

We supplement certificate error reports with a separate dataset of Chrome telemetry data. Telemetry data includes pseudonymous counts of browser events. We rely on telemetry data to corroborate patterns in our main dataset and analyze data that is not included in the certificate reports.

The telemetry and certificate upload services differ in two important ways. First, telemetry reports are queued and retried every five to thirty minutes (depending on the operating system and network type) if an attempt to send them fails. This means that we will reliably receive telemetry data — but not error reports — from clients with flaky network connections. Second, telemetry reports

[2] We recently implemented a retry feature. Since this feature was not present for most of the data analyzed in this work, we exclude reports sent via retry from the analysis.
[3] Examples include Iridum Browser (https://iridiumbrowser.de) and Amigo (https://amigo.mail.ru)

are sent over HTTPS, so we do not receive telemetry data from clients that are persistently unable to send HTTPS requests.

We occasionally refer to Chrome telemetry data to study events that are not captured by our main dataset of certificate reports. For example, certificate reports are not sent in cases where users do not encounter certificate warnings, so we use telemetry data to investigate HTTPS-related events that do not trigger certificate warnings.

4 ERROR DEFINITIONS AND CLASSIFICATION METHODS

In this section, we define the different types of errors, give background on their causes, and describe our classification rules.

To initiate the project, a group of browser security experts manually investigated and labeled more than 2,000 reports over several months. In some cases, the error causes were obvious; in others, it required research into network appliances and consumer software. Based on our review experiences, we wrote rules to automatically classify reports. A daily analysis pipeline parses incoming certificate error reports and applies our rules. Our goal is to assign blame (server, network, or client misconfiguration) and a specific root cause. Each report can contain multiple certificate validation errors, and we attempt to assign blame and cause for all errors.

4.1 Classifying server errors

A *server error* occurs when a server presents an invalid or incomplete certificate chain. A properly configured client on a properly configured network should be able to validate a server's certificate chain. If it cannot, then we blame the server for the error. An example is a server that presents a self-signed certificate.

When processing a report, we check to see whether the Googlebot (Google's web crawler) has encountered any certificate errors for that website within the past thirty days. The Googlebot serves as ground truth: it is a properly configured client on a properly configured network, and it should be able to validate a server's certificate chain. If the Googlebot has seen a matching error for a website, then we blame the server. Googlebot does not necessarily crawl every site every day, so we use a thirty day window rather than a same-day window to increase the chance that the Googlebot has crawled a particular site within the window.

Note that error reports include the server-supplied chain. Why don't we use that instead of the Googlebot? Theoretically, we could simply check whether the server-supplied chain validates. In practice, server-supplied chains are unreliable due to TLS proxies. Reports often contain certificates generated and signed by proxies, and proxies often introduce certificate errors [10]. To avoid mixing server errors and network errors, we use the Googlebot to tell us what certificate chain the server was sending around that date. One potential concern is that the Googlebot might validate certificate chains differently from clients. To verify our methodological choice, we sampled 2,296,747 certificate chains from the Googlebot and re-validated them in Ubuntu and Windows. The three platforms agreed whether a chain should validate 99.87% of the time.

Beyond placing blame on the server, we further categorize server errors by the specific type of misconfiguration:

4.1.1 *Server date errors.*

Certificates are only valid within a certain date range. A *server date error* occurs when a server uses a certificate prematurely or past its expiration date. If a client reports that a certificate was not yet valid or expired, then we check whether the Googlebot encountered the same problem on the reported website in the previous thirty days. If so, we classify the error as caused by a server date error.

4.1.2 *Server name-mismatch errors.*

Certificates are only valid for the hostnames listed in the certificate. The hostnames must be listed precisely or with a wildcard (e.g., `*.example.com`). A *server name-mismatch error* occurs when a server deploys a certificate without including the website's hostname or matching wildcard. If a client reports that a certificate is missing a hostname, we check whether the Googlebot encountered the same problem on the website in the previous thirty days. If so, we classify the error as caused by a server name mismatch.

To understand why the error occurred, we further look for two developer mistakes that can lead to name mismatch errors, as previously identified by Akhawe et al [9]. We look for two types of subdomain mismanagement: *www mismatch error* and *out-of-wildcard-scope subdomain error.* A www mismatch error occurs when a client tries to visit `example.com` but gets a certificate for `www.example.com` (or the other way around). An out-of-wildcard-scope subdomain error occurs when a client tries to visit `a.b.example.com` but gets a certificate for `*.example.com`; this fails because wildcards only match a single DNS label level.

4.1.3 *Server authority-invalid errors.*

Certificates are only valid if they chain to a trusted root. A *server authority-invalid error* occurs when a server deploys a certificate that does not chain to a trusted root (for example, a self-signed certificate). If a client reports that a certificate doesn't chain to a trusted root, then we check whether the Googlebot encountered the same problem on the reported website in the prior thirty days. If so, we classify the error as caused by a server authority-invalid error. We further identify *self-signed certificates* as a sub-category of server authority-invalid errors.

One notable decision relates to how we classify errors caused by untrusted government-operated roots. Some government websites use government-operated roots that are not included in standard root trust stores. Citizens of these countries are expected to install these roots on their devices, but in practice many people do not. Should we blame the server for using a non-standard root, or should we blame clients for not installing the root? We choose to designate such errors as server errors.

Our classification misses one category of server authority-invalid errors: errors on intranet websites. The Googlebot cannot reach intranet websites for classification. Since we cannot differentiate between server, client, and network errors for intranet websites, we leave them as unclassified.

4.1.4 *Server insufficient-intermediates errors.*

Servers are supposed to provide enough information for a client to build a full chain from the leaf certificate to the trusted root certificate. Typically, servers must provide intermediate certificates between the leaf and root. A *server insufficient-intermediates error* occurs when a client can't build a valid chain because the server didn't include all of the necessary intermediate certificates.

Insufficient intermediate errors are tricky for two reasons. First, they are context-dependent. If a client happens to have a missing intermediate cached (from a previous website), or if the client actively fetches the missing intermediate, then the chain will appear valid. Second, they look similar to server authority-invalid errors: in both cases, the client can't build a chain to a trusted root. The distinction is that chains with insufficient intermediates *would* validate on most clients if the server provided more information.

If a client reports that a certificate doesn't chain to a trusted root, we perform two steps. First, we check whether the Googlebot encountered the same problem on the reported website in the prior thirty days. The Googlebot caches intermediates, so websites with insufficient intermediate errors usually look error-free to the Googlebot. This first check filters out server authority-invalid errors. If the certificate chain looks valid to the Googlebot, we then attempt to build a chain using only the certificates that the server supplied to the Googlebot. If the resulting chain doesn't validate, then we classify the error as caused by an insufficient intermediate error.

This heuristic can have false positives or false negatives:

- A false positive happens if the client has a missing intermediate cached (which would allow the chain to validate), but something else coincidentally went wrong. The server really is missing an intermediate but it was not the cause of the error. In July 2016, we manually reviewed 100 reports classified as insufficient intermediates and found one false positive report.
- A false negative happens if the server-supplied certificates chain to a root that the Googlebot trusts but the client does not. For example, a client might have an older trust store that does not include a newer root that Googlebot trusts. The server might send a chain that is recognized by the newer trust store, but if the server neglects to send a cross-sign certificate linking the newer root to an older root in the client's trust store, then the client will be unable to validate the chain unless it happens to have the cross-sign certificate cached. In this case, we will leave the report as unclassified. We observed 5 false negatives when manually reviewing 100 unclassified reports (Section 9).

We classify insufficient intermediates as a server problem because servers are supposed to supply intermediates (as per RFC 5246 [11]). However, one could argue that they are a client problem because clients can dynamically fetch intermediate certificates. Some web browsers already do this as needed. Was the server misconfigured because it didn't send the intermediates, or was the client misconfigured because it didn't fetch them? We label insufficient intermediates as server errors because they violate the HTTPS specification, but the alternate perspective is also reasonable.

4.1.5 Server SHA-1 errors. Certificates signed with the outdated SHA-1 hash algorithm are no longer considered secure, and Chrome has gradually phased out support for SHA-1 over the past few years [17]. As of Chrome 57, users see certificate errors for any site with a SHA-1 signature in its certificate chain. We do not include SHA-1 errors in our automated analysis pipeline or in the bulk of our analysis because Chrome's SHA-1 support changed over the course of our dataset, and the algorithm was not fully blocked until nearly the end of the period that we studied. However, we include an analysis of SHA-1 errors on a recent subset of the data in Section 6.2.5.

4.2 Classifying client errors

A *client error* occurs when a client cannot validate a certificate chain from a properly configured server. A properly configured client would be able to validate the same certificate chain. We identify the following types of client errors:

4.2.1 Incorrect client clocks. Certificates are only valid within a certain date range. A *client clock error* occurs when a client's clock is set too far in the future or past, causing certificates to look as if they are outside of their validity periods. If a certificate date error was not caused by a server misconfiguration, we next check whether the reported certificate chain's dates are valid relative to our own server's clock. If it is, then we classify the error as caused by a client clock error.

4.2.2 Anti-virus errors. Anti-virus (AV) software commonly acts as a TLS proxy in order to inspect HTTPS browser traffic. An *anti-virus error* occurs when a bug in an AV proxy prevents clients from establishing valid HTTPS connections. During the course of manual review, we observed several instances of AV errors. When a certificate report contains the name of a popular AV product (Avast, Kaspersky, Bitdefender, or Sophos) in the certificate chain, the pipeline flags the report as an anti-virus report.

We do not use the AV label to automatically assign a root cause. AV product names appear in many reports that have other error causes. Each AV bug has had its own distinct signature, which has prevented us from writing a single rule that captures this error class. Instead, we monitor the number of AV-related reports. If there is an upswing, we manually investigate the situation. Several times, these upswings have turned out to be bugs in AV products (Section 7.2).

4.3 Classifying network errors

A *network error* occurs when a network appliance intercepts an HTTPS connection and replaces the certificate chain with one that the client cannot validate. Our pipeline classifies the following types of network errors:

4.3.1 Captive portal errors. Airport, hotel, and enterprise networks often block access to the Internet until the user has authenticated. Network access points that enforce this requirement are known as captive portals. A *captive portal error* occurs when a captive portal intercepts TCP packets or DNS queries to redirect HTTPS traffic to the captive portal's login page. This behavior causes a name mismatch error because the hostname that the browser requested does not match the certificate presented by the login page.

When a Chrome user encounters a name mismatch error, Chrome sends a probe request to an endpoint with a known response. If the response is unexpected (as it would be when redirected to a captive portal login page), then Chrome prompts the user to log in to the captive portal. Certificate reports contain a flag to indicate whether such a prompt was shown.

The pipeline does *not* classify an error as a captive portal error solely because the report says that Chrome detected a captive portal.

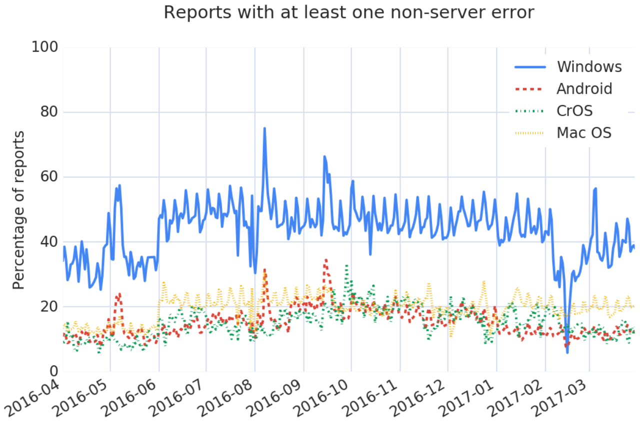


Figure 2: Percentage of reports with at least one error caused by the client or network, broken down by platform.[4]

We found that the captive portal probe has a high false positive rate, which we discuss further in Section 8.1.

Instead, the pipeline produces a weekly list of the most common certificate chains in captive portal error reports. We manually curate them to maintain an ongoing list of the common captive portal vendors. Whenever a name mismatch error appears for one of these known captive portal certificates, the pipeline marks the error as caused by a captive portal. Unfortunately, the ongoing expansion of the captive portal list makes it difficult to compare the frequency of captive portal errors across time because we do not re-label reports retroactively after new captive portals have been identified.

4.3.2 Missing TLS proxy roots. Enterprises, schools, and other entities commonly install network middleboxes that intercept TLS connections. Devices on these networks are expected to have the middlebox's root certificate installed, but in practice, this is not always the case. A *missing TLS proxy root error* occurs when a user skips installing the root and tries to visit a website that is intercepted by the proxy.

When an authority invalid error does not appear to be a server error, we compare the reported certificate chain against a list of TLS proxy products. We compiled this list by manually inspecting a sample of certificate reports. The list includes: Fortigate/Fortinet, Cyberoam, Cisco Umbrella, Bluecoat, and McAfee Web Gateway. If the name of any of these vendors appears in the certificate chain, we classify the error as caused by a missing TLS proxy root.

5 OVERVIEW OF RESULTS

This section provides an overview of our dataset and main findings. In Sections 6-9, we discuss the most common server, client, and network errors in more detail.

We can automatically classify two-thirds of reports. Our analysis pipeline assigned at least one root cause to 62.8% of the Annual Reports sample. The impact of each root cause is shown in Table 1.

Client and network errors play a large role in HTTPS error warnings. Prior work emphasized the role of developers in HTTPS errors [9, 12, 14, 18]. However, we find that client and network problems are at least as influential as server misconfigurations. Of the reports that we can automatically classify, half are server errors (31.2% of all reports) and half are client or network errors (31.6% of all reports). Per a manual review, the unclassified reports are even more weighted towards non-server errors (Section 9). Figure 2 shows the percentage of all reports with at least one labeled client or network error, as per the analysis pipeline. Client and network misconfigurations are more problematic for Chrome users on Windows than on other platforms, primarily due to the prevalence of misconfigured client clocks on Windows (Section 7.1).

A small number of root causes account for a large amount of spurious warnings. If the most common root causes could be addressed, a large chunk of spurious warnings would disappear. For Windows, client clock errors account for more than 30% of all certificate warnings. Similarly, on Android, insufficient intermediates cause more than 35% of certificate warnings. We therefore target these influential errors when building mitigations (Section 10).

We also find that government websites are disproportionately responsible for server errors. Fixing this is beyond the scope of our work, but we urge citizens to voice concerns.

6 SERVER ERRORS

We want to answer two research questions: (1) Are some types of sites more prone to server errors? (2) Are some types of server errors more common than others?

To answer these questions, we take a closer look at the reports that our pipeline labeled as server errors.

6.1 Types of sites with server errors

Government-run websites with server errors are responsible for a disproportionate number of HTTPS errors. We selected the 100 sites with the most server error reports in the Annual Reports sample and manually assigned category labels to them. To obtain this list of sites, we grouped reports by their hostname, and then took the 100 hostnames with the most reports classified by our analysis pipeline as caused by a server error. Table 2 shows this top 100 by category. 65% of the "worst offenders" are government-run websites. They exhibit a range of misconfigurations, ranging from hostname mismatches to revoked certificates.

During this time period, we reached out to three national governments to notify them of the problem. The U.S. government was responsive and fixed misconfigurations on several dozen sites.

6.2 Types of common server errors

Table 1 shows the frequency of different types of server errors. Insufficient intermediates are the leading cause of server errors, and they are primarily seen on Android.

6.2.1 Insufficient intermediates. Insufficient intermediates are a large problem on Android, where they are responsible for 36% of HTTPS error reports. This class of error is much less common on other operating systems. Chrome relies on the operating system for

[4] The February 2017 dip in non-server errors on Windows is due to a Chrome bug that was pulled into a Chromium fork at that time. The bug triggered warnings on many popular websites that were not classified by our pipeline.

Table 1: Percentage of reports with each root cause in the Annual Reports sample. A report can have multiple errors and root causes. Each cell is the percentage of the total Annual Reports sample that was labeled with the row's root cause.

	Windows	Mac OS	ChromeOS	Android
Server errors				
Insufficient intermediates	1.26%	4.80%	0.783%	35.8%
Authority invalid	6.11%	5.54%	3.49%	6.01%
Name mismatch	11.7%	11.6%	7.59 %	9.77%
Date error	4.23%	4.39%	2.80%	2.73%
Client errors				
Incorrect system clock	33.5%	8.71%	1.72%	8.46%
Network errors				
Captive portal	0.925%	5.46%	4.57%	2.11%
Missing TLS proxy root	6.57%	3.34%	9.13%	1.16%

Table 2: Breakdown of the 100 "worst offenders": the sites with server errors that generated the most reports.

Category	Number of sites
Government	65
Education	7
Email	5
Malware-associated advertising	4
Finance	4
E-commerce	3
File-sharing	3
Telecommunications	3
Other	6

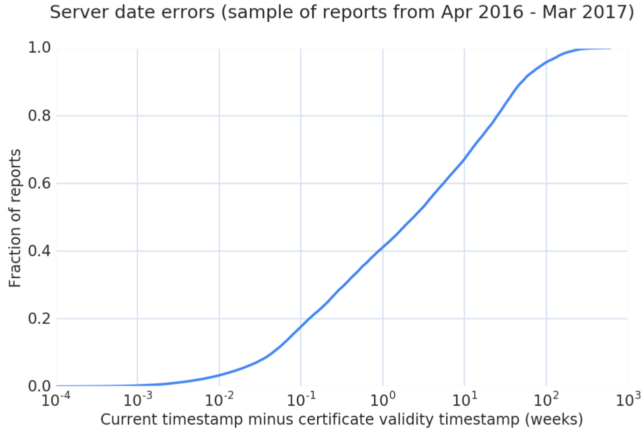


Figure 3: CDF of the amount of time by which certificates are expired. X-axis is log scale.

certificate validation, and Android doesn't use *authorityInfoAccess (AIA) fetching* to dynamically fetch intermediates while verifying the certificate. Windows, Mac, and Chrome OS do implement AIA fetching. Sites whose certificates validate in Chrome on other operating systems will fail to validate on Android unless the device already has the intermediates cached from prior connections to other websites. This is likely also a problem in Mozilla Firefox, which doesn't perform AIA fetching on any operating system.

These misconfigurations fall into two categories:

(1) The server sends only a single leaf certificate.
(2) The server sends some intermediates, but they are the wrong intermediates or not the full set. For example, the server might send intermediates that chain to a root that is not widely trusted, but neglect to send a cross-sign certificate that chains that root to a more widely trusted root [4].

The first category – a single leaf certificate – dominates. The server sent only a single certificate in 87.3% of the insufficient intermediate errors from the Annual Reports sample. This suggests that the problem is largely caused by server operators that are not aware that they should serve intermediates along with the leaf, or don't know how.

6.2.2 Name mismatch errors. Akhawe et al. previously found that name mismatches are commonly due to subdomain mismanagement [9]. We consider two cases that are of interest: www mismatches and out-of-wildcard-scope subdomains, both of which are defined in Section 4.1.2. In the Annual Reports sample, 3.7% of server name-mismatch errors were www mismatches, and 13.2% were out-of-wildcard-scope subdomains.

6.2.3 Authority invalid errors. In the Annual Reports sample, 18.3% of server authority-invalid errors (excluding those caused by insufficient intermediates) were for self-signed certificates.

We also consider the proportion of errors that occur on intranet hosts. We generally do not classify errors for intranet hosts as server errors because the Googlebot cannot contact these hosts to determine whether the server is properly configured. However, 5.3% of all authority invalid errors during this time period were for intranet hostnames or non-routable IP addresses. We therefore suspect that intranet hosts with invalid certificates are a common source of certificate errors, but it is difficult to say for sure because some of this 5.3% could have been caused by client or network misconfigurations (e.g., misconfigured corporate middleboxes).

6.2.4 Server date errors. Virtually all server date errors are caused by expired certificates, rather than certificates that are not yet valid. Figure 3 shows the distribution of these expired certificates relative to the time at which a report was received. Notably, 57% of certificate reports were expired by less than 30 days, and 75% by less than 120 days.

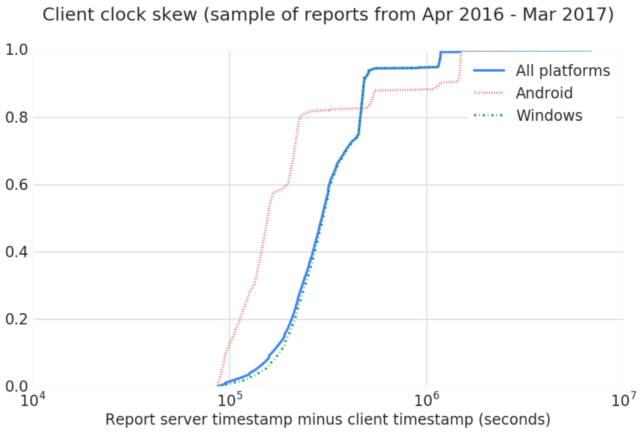


Figure 4: CDF of client clock skews for client clocks that are in the past, from a sample of 100,000 certificate reports. We only include reports where the client clock was at least 24 hours behind and no more than 3 months behind. X-axis is log scale.

6.2.5 Server SHA-1 errors. As discussed in Section 4.1.5, our analysis pipeline does not automatically classify SHA-1 errors because Chrome did not fully block SHA-1 certificates until near the end of the Annual Reports time period. We retroactively identified server SHA-1 errors from the February - March, 2017 subset of the Annual Reports sample, covering Chrome 56 and 57. Chrome 56 removed SHA-1 support for publicly trusted roots and was released to the stable channel in late January, 2017. Chrome 57 was released in mid-March and removed SHA-1 support for all certificate chains, including locally installed roots. During this time period, server SHA-1 errors accounted for 9.4% of all certificate reports. By June 2017 (Chrome 59), server SHA-1 errors declined to 2.7% of all reports.

7 CLIENT ERRORS

Various conditions on end-user machines can cause spurious HTTPS warnings. These misconfigurations can result from seemingly innocuous changes to system settings, or from users installing malware, debugging tools, and even security products. In this section we discuss the most problematic client misconfigurations.

7.1 Client clocks

Client clock problems are widespread, particularly on Windows. Table 1 shows the relative frequency of clock errors.

How far off are client clocks? Nearly all incorrect client clocks in our dataset are in the near past. In the Annual Reports sample, the client clock was more than 24 hours behind in 6.7% of reports and more than 24 hours ahead in 0.05% of reports. Of reports where the client clock was off by more than 24 hours in either direction, 99.8% were within 3 months of true time. Figure 4 shows the CDF of the client clock skew from a random sample of 100,000 reports.

Why are clocks so frequently misconfigured? We are unsure. Users might manually set their clocks incorrectly to get around software licensing restrictions or to cheat in games. Incorrect system clocks are a common complaint associated with malware on online help forums [5, 25], but the phenomenon of malware interfering with system clocks is not, to our knowledge, well-studied. We also suspect that some incorrect client clocks are due to hardware issues like dying CMOS batteries.

Site owners need to manage certificates carefully to avoid breakage due to client clocks. Figure 2 shows a spike in client-caused errors on Windows in September 2016. This spike corresponded to a short-notice rollout of newly issued Google certificates. Certificates that are used close to their issuance date fall afoul of misconfigured client clocks more often because the clock is more likely to fall before the certificate's validity period begins.

7.2 Anti-virus

Consumer anti-virus products commonly install root certificates and intercept TLS connections to look for suspicious traffic [6, 26]. Bugs in their TLS interception may cause HTTPS error warnings. We have observed high-impact instances of these bugs.

Though touted as a security feature, the practice of TLS interception has numerous downsides from a security perspective:

- TLS interception opens the door for misconfigurations that cause spurious certificate warnings. Even small logic bugs can cause HTTPS errors. These bugs and misconfigurations often have the unfortunate property that they affect every single HTTPS site that a user visits, even stymying a user's ability to search for help.
- When a locally installed root of trust is in use, Chrome disables various certificate validation and TLS security checks that cannot reasonably be enforced for local roots, such as HTTP Public Key Pinning [7]. Most proxies do not implement these checks themselves, and in many cases do not perform even basic certificate validation, leaving the user vulnerable to attack [13, 24].

As one example of a high-impact anti-virus issue, we discovered a bug in Avast that temporarily caused widespread HTTPS error warnings. In September 2015, we noticed a large number of reports for certificate date errors on properly configured sites. The certificates in the reports chained to expired Avast roots. We estimated an impact of about 1.5 million certificate warnings per week. The cause was that Avast's software was generating a root certificate on installation using the system clock's time. If the clock was wrong at installation time and then later corrected, Avast would continue to intercept TLS connections with an expired root certificate. This led Avast users to see HTTPS error warnings on every site they visited. We reported the bug to Avast and they quickly pushed an update that fixed the bug by querying a server for an accurate timestamp to use when generating a root certificate.

Another example is that anti-virus products might generate certificates that were once valid but are no longer accepted by Chrome, either because the anti-virus product is out-of-date or because its maintainers are not keeping up with best practices. For example, in a manual inspection of unclassified reports (Section 9), we encountered anti-virus products that use SHA-1 signatures. SHA-1 signatures are no longer considered valid in Chrome.

8 NETWORK ERRORS

Network errors are an especially interesting and challenging class of error. HTTPS is designed to prevent network actors from intercepting HTTPS, yet there are several common use cases where network appliances attempt to do this. Whether this interception is benign or malicious depends on how much the user trusts the network appliance owner and his/her intentions.

8.1 Captive portals

Captive portals cause name mismatch errors when users first connect to a network that requires authentication. We find that captive portals are one of the smaller error causes, but we are likely undercounting them because they are difficult to automatically identify.

On some operating systems, Chrome uses a standard captive portal detection technique of sending network probes. These operating systems are Windows 7 and below, Mac OS, Chrome OS, and Linux. On other OSes, Chrome relies on the system's captive portal detection to detect the portal and prompt the user to log in. We find that Chrome's technique suffers from both false positives and false negatives:

- A false positive occurs when Chrome's probe request detects a captive portal but there was no captive portal. We manually reviewed 100 reports from May 2017 and found 34 false positives. We attribute most of them to home routers and enterprise middleboxes, which interfere with the probe request despite not being traditional captive portals.
- A false negative occurs when Chrome's probe request fails to detect a captive portal. Of the captive portal errors caught by our human-curated rules (described in Section 4.3.1), 30.1% were not identified by Chrome. We attribute the high false negative rate to slow captive portals. According to Chrome telemetry, only 54% of captive portal probe requests respond within 3 seconds, which is the maximum amount of time that Chrome will wait for a probe to respond before drawing error UI. Moreover, a preliminary survey of captive portals in Japan suggest that some portals intentionally evade detection for unknown reasons [20].

Our findings show that using network probes to detect captive portals is difficult and unreliable. In addition, we believe that we are missing error reports from many captive portals. A captive portal typically blocks reports from being sent until the user has logged in to the portal. Since Chrome did not retry failed report uploads until very recently, we do not expect to receive reports that were blocked by captive portals. We only receive such reports when the portal does not block the upload for some reason, or when the user authenticates with the portal before the certificate warning is dismissed. Therefore, we suspect that the fraction of certificate errors caused by captive portals, as shown in Table 1, significantly undercounts the problems that they cause.

Of the captive portal reports that we do receive, interestingly, a large number of them share the same few certificate chains from a handful of captive portal vendors, as shown in Table 3. This gives some hope that if a small number of vendors adopted better captive portal implementations that did not cause spurious certificate warnings, the problem could be significantly alleviated.

Table 3: The top five most common captive portal vendors in certificate reports from Sept 13 - Oct 10 2016.

Captive portal vendor	Percent of all reports
Aruba Networks	0.95%
Orange France	0.14%
AlwaysOn	0.12%
GlobalSuite	0.09%
AccessNetwork.ru	0.09%

8.2 TLS proxying

Enterprises, schools, and even home networks often have middleboxes that intercept TLS connections using their own root certificates, which are intended to be installed on devices on the network. These middleboxes introduce many of the same security problems that consumer anti-virus introduces. As discussed in Section 7.2, TLS proxies on both the client and the network override Chrome's security checks and can introduce bugs that cause error warnings.

We find that missing roots for network middleboxes are a widespread problem. Our pipeline classifies such errors by looking for several popular middlebox product names in the certificate chain (as described in Section 4.3.2). Table 1 shows the relative frequency of this error class. Our classification is conservative; when we reviewed unclassified reports (Section 9), we found that many are due to other TLS proxy products not covered by our rules.

Missing root certificates cause the vast majority of certificate errors that users of these products encounter. For each of these products, more than 80% of certificate errors chaining to the product's certificate were caused by a missing root. When a user of one of these products sees a certificate error, it is very likely to be due to a missing root rather than any other cause.

In addition to missing roots, we observe by manual review that TLS proxies introduce spurious certificate warnings by means of other misconfigurations as well. For example, some middleboxes use SHA-1 signatures, which Chrome no longer accepts as valid.

9 UNCLASSIFIED ERRORS

Our analysis pipeline does not automatically assign a root cause for 37% of reports. To characterize the unclassified reports, we manually reviewed a random sample of 100 unclassified reports from May 2017. (We reviewed recent reports that had not yet been stripped of details, since otherwise it would be difficult to investigate the cause of a report.) Table 4 shows the results. When the pipeline does not automatically assign a root cause, it is often because the report is for a site about which Googlebot has no data (e.g. an intranet site) or because the error was caused by a TLS proxy or captive portal that our pipeline does not look for. As described in Section 4.1.5, our pipeline does not yet attempt to assign root cause for certificate warnings that are due to SHA-1 signatures, because Chrome had not yet fully removed SHA-1 support during most of our dataset.

Our manual analysis revealed more client and network misconfigurations than server misconfigurations. We anticipated this finding because our automatic analysis shows an even breakdown between client/network and server misconfigurations, but with a known under-count of captive portal errors.

Table 4: Manually assigned root causes from a random sample of 100 reports in May 2017 for which our analysis pipeline did not automatically assign a root cause.

Error cause from manual inspection	Count
Server errors	
Server certificate uses weak signature algorithm	10
Server certificate has a name mismatch	9
Insufficient intermediates	5
Government root certificate that isn't widely trusted	5
Intranet IP without valid cert	4
Server certificate chains to distrusted root	2
Server certificate has multiple errors	2
Total	37
Network errors	
Captive portal	22
Corporate middlebox	8
School middlebox	7
Misconfigured home router	4
Other middlebox	3
Total	44
Client errors	
Old or corrupted root store	2
Ad blocker or anti-virus using weak signatures	2
Expired anti-virus root	1
Local server	1
Incorrect clock	1
Total	7
Unknown	12

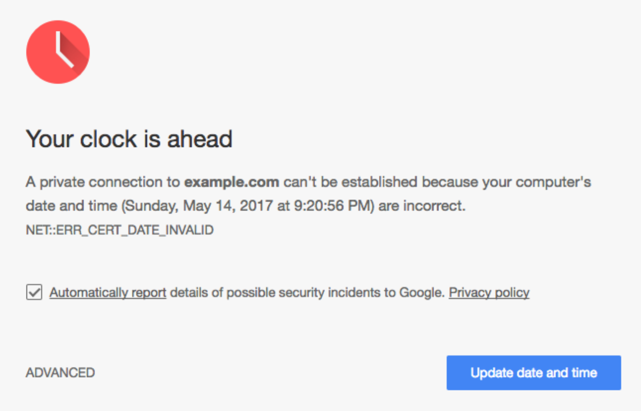
10 MITIGATIONS

Our ultimate goal is to stop showing unnecessary HTTPS error warnings. In this section, we propose, discuss, and evaluate mitigations for many of the misconfigurations that cause Chrome users to see unnecessary warnings. We expect that these mitigations will or have already replaced about 25% of certificate errors in Chrome.

When possible, we would like to avoid showing any error UI at all. Ideally, browsers would be able to automatically correct or work around the misconfiguration in a way that is invisible to the end user. When that isn't possible, we aim to replace certificate warnings with actionable, non-scary explanations of the error. These explanations should pinpoint the cause of the error and prompt the user to fix it. Both approaches require caution because attackers can make their attacks look like misconfigurations. We therefore must ensure that our mitigations are not advantageous to attackers.

10.1 Stopping client clock errors

We tackled client clock errors by implementing a special warning to show when the user's clock is wrong (Figure 5). To prevent attackers from leveraging this less-scary UI, users cannot click through – they have to fix their clocks to get to the site. We built this warning by using a heuristic based on the build time to guess when the clock is wrong. We then used our dataset of certificate reports to investigate the effectiveness of the heuristic. Since its performance was not satisfactory, we built and evaluated a replacement secure time service.



Your clock is ahead

A private connection to **example.com** can't be established because your computer's date and time (Sunday, May 14, 2017 at 9:20:56 PM) are incorrect.

NET::ERR_CERT_DATE_INVALID

☑ Automatically report details of possible security incidents to Google. Privacy policy

Figure 5: The UI that Chrome shows when it detects that a certificate error is caused by a client clock error.

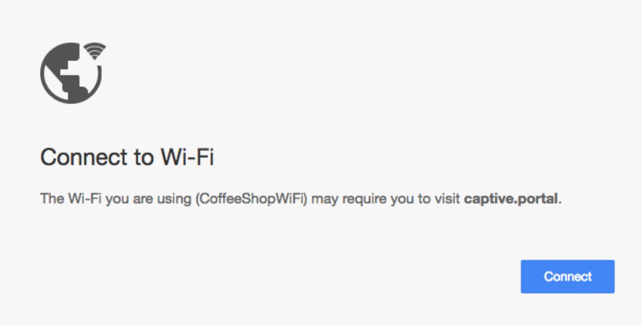
10.1.1 Build time heuristic. Chrome's build time heuristic compares the current system time to the binary build time. If the system time is either one year behind the build timestamp or more than two days ahead of it, then Chrome will show the clock warning when it encounters a certificate date error.

We find that the build time heuristic has many false negatives. We are able to evaluate it post hoc by looking at the client time in the reports. From April 30 to May 13, 2017, the heuristic only detected 68% of certificate errors that were caused by incorrect client clocks. For the remainder, Chrome showed the generic certificate warning. This suggested to us that we needed to improve client clock detection beyond the build time heuristic.

When the heuristic is able to identify a client clock error, the UI (Figure 5) proves helpful. In 53% of reports associated with this UI, the user changed their clock (by at least 6 hours) before the warning was dismissed. This compares to 3.9% of the time when Chrome showed a generic certificate warning for client clock errors. We interpret this to mean that actionable errors are, in fact, more helpful than generic security warnings.

10.1.2 Secure time. To improve client clock error detection, we implemented a secure time service that Chrome queries when it encounters a certificate date error. Upon encountering a certificate with invalid dates, Chrome queries an update server for the current time (an HTTP URL, with the response signed by the private key corresponding to a public key baked into Chrome), and delays showing a warning for up to three seconds. If the query returns within three seconds and indicates a timestamp that is significantly skewed from the local system clock, then Chrome shows the clock warning from Figure 5.

An analysis of certificate reports from an experimental launch shows that the secure time service improves detection of client clock errors to 96%, with 93% of queries to the time service completing in under three seconds. (Even if a query does not complete within three seconds, the result — once eventually received — will be cached



Connect to Wi-Fi

The Wi-Fi you are using (CoffeeShopWiFi) may require you to visit **captive.portal**.

Figure 6: The warning that Chrome shows when it detects that a certificate error is caused by a captive portal.

for use if subsequent certificate date errors are encountered.) The secure time feature launched to Chrome stable in May 2017.

10.1.3 Future work. We would ultimately like to invisibly correct client clock errors. To do this, Chrome would need to use the timestamp fetched from the secure time service for all certificate validations. This would stop the errors without needing any error UI. However, the challenges in doing so are twofold:

(1) Chrome relies on the platform's certificate validation library. On some platforms, it is not possible to provide a time other than the system time as input to certificate validation. Using a timestamp from the secure time service would require Chrome to implement its own certificate validation.
(2) Even though Chrome could invisibly correct the misconfiguration by using the timestamp fetched from the secure time service for certificate validation, it might still be desirable to alert the user to the problem so that they can fix the system clock. Other applications on the user's device might be functioning incorrectly because of the incorrect system time. We might want to devise some way to prompt the user to fix their system clock without interrupting their normal browsing as the clock error UI currently does.

Another area of future work is to reduce trust in Chrome's update server. Instead of using an update server as the secure time service, Chrome could implement a protocol such as Roughtime [3] for secure decentralized time synchronization.

10.2 Captive portal detection

Chrome sends network probes to attempt to detect captive portals. If a captive portal is detected, Chrome displays a special captive portal error UI (as shown in Figure 6) instead of a security warning. We consider this actionable message to be a significantly better user experience than showing the generic certificate warning. However, we found that the probe request has high false positive and false negative rates (Section 8.1).

Industry standards groups are working on improved solutions for captive portal discovery [1], but these solutions will take time to develop and roll out. In the meantime, we implemented two improvements to Chrome's captive portal logic:

10.2.1 Shipping known captive portals. Our analysis pipeline produces a list of candidate captive portal certificates, which we manually curate each week to produce a list of known captive portals (Section 8.1). By shipping this list in Chrome and using it to supplement captive portal detection, we would nearly double the detection rate of certificate errors caused by captive portals. We implemented this by putting the captive portal list in a dynamically updateable Chrome component, so that additions to the list can be shipped to clients on an ongoing basis (without being tied to the release cycle). We launched this feature as an experiment on Chrome's canary and dev channels. Telemetry from this experiment shows that 3.8% of name mismatch errors match a captive portal certificate on the list, and we plan to expand the experiment.

10.2.2 Certificate report retry. We would like to retry report uploads, similar to Chrome's telemetry system, to get better visibility into captive portals. Retrying would allow us to receive reports caused by captive portals even if the captive portal prevents them from being sent until after the user authenticates with the portal.

We implemented report retries using the same logic that Chrome's telemetry system uses, though we maintain reports in memory only and do not persist them to disk. The implementation is not yet widely deployed enough to report results, but we hope to gain a more accurate picture of certificate errors caused by captive portals. It should also help us expand our list of known captive portals.

10.3 AIA fetching on Android

Insufficient intermediates are a large problem on Chrome for Android, accounting for 36% of all certificate warnings on Android. This happens because the platform does not fetch intermediates as other platforms do during certificate validation.

To work around this, we implemented AIA fetching in Chrome for Android. When the platform certificate verifier returns an authority invalid error, Chrome looks at the last certificate for which there is no issuer in the server-sent chain. If this certificate has an AIA URL, Chrome fetches it and again attempts a platform certificate verification. If it again fails, Chrome repeats the process, until a valid certificate chain has been found or until exhausting a maximum number of fetches.

AIA fetching is implemented in Chrome 58. Since this feature launched, the percentage of certificate errors caused by missing intermediates on Android has steadily declined to 3.0% as of August 2017. The remaining errors are likely due to network flakiness, which could potentially be improved by retrying failed AIA fetches.

10.3.1 Future work. If the Android certificate verifier directly supported AIA fetching, then it would likely be more performant than implementing it in Chrome. Android support for AIA fetching would also benefit other Android applications besides Chrome. However, Android update cycles are much slower than Chrome's, so Chrome on Android is likely to need to support AIA fetching for the foreseeable future.

10.4 Redirecting for related name mismatches

Name mismatch errors account for a notable fraction of errors. We would like for the browser to handle this class of error automatically. The core idea is to redirect the user to the domain with a valid certificate, if (a) the redirection is safe and (b) we think that is likely where the user wants to go. We must be cautious with redirections because different subdomains can be controlled by different people.

We decided to start with "www mismatch" errors, which are responsible for a small percentage of certificate name mismatches (Section 6.2.2). We felt that the risk was low given some browser display logic already assumes that the www subdomain and TLD are operated by the same party. When Chrome encounters a name mismatch error for `www.example.com`, it issues a background request to `example.com` (or the other way around). If the background request responds within three seconds with an HTTP 200 status code, then Chrome redirects the user there. We also place a message in the developer console to alert the site owner to the misconfiguration.

Chrome telemetry data shows that this redirection occurs for 1.8% of all name mismatch errors. A www mismatch is found but the redirect URL is not available for 0.28% of name mismatch errors. Although 1.8% is small, we consider this a success. We are considering expanding the redirection to include other sets of subdomains, such as redirecting `m.example.com` to `example.com`.

10.5 Future mitigations

To continue tackling causes of spurious certificate warnings, we are planning to explore several mitigations and research directions.

10.5.1 Government roots. Our manual classification (Section 9) finds that servers commonly use government root certificates that are not widely trusted by clients. When Chrome encounters an authority invalid error for a certificate that chains to a known government root, the warning UI could direct the user to a webpage that explains what the government root is and how to install it.

The primary challenge in implementing this mitigation is the messaging and UX. Installing a government root certificate can be risky for some users – for example, if they don't trust the government in question, or if the government does not operate its root in accordance with industry standards. Chrome should provide information about how to fix the error without encouraging users to install a root that they might not fully trust.

10.5.2 TLS proxy roots. In Section 8.2, we noted that a significant fraction of errors are caused by a small number of TLS proxy products. Chrome could look for these product names in the issuer strings of certificates that generate errors. However, it is unclear what Chrome should do if it detects that an error is possibly due to a missing TLS proxy root. One option would be to prompt the user to contact a network administrator.

10.5.3 Outreach for misconfigured servers. Prior work has investigated the effectiveness of notifying site owners about web server hijacking [21]. Similar studies could be undertaken for HTTPS misconfigurations. For example:

- Are site owners more likely to correct misconfigurations if they receive email messages about them, rather than just browser warnings?
- Are email messages more effective if they contain an estimate of the number of warnings Chrome has shown for the site?
- Are email messages more effective if they include instructions about how to fix the misconfiguration?

11 RELATED WORK

In this section, we survey other studies of HTTPS errors and misconfigurations and compare them to our work.

11.1 Studies of HTTPS errors

Our dataset differs from prior work because it includes non-server errors and a global perspective; further, we deploy solutions.

11.1.1 Warnings in the field. Akhawe et al. were the first to study the causes of HTTPS warnings in the field [9]. They monitored network traffic from the egress points of ten U.S. research, government, and university networks. Their study had a population of 300,000 users over a nine-month period. They identified self-signed certificates, expired certificates, name mismatches related to subdomains, and incomplete chains as causes of TLS errors.

These prior findings did not encompass our experience working in support forums, where people commonly report issues due to client misconfigurations and network interference. We were therefore concerned that the study's results were only part of the picture, due to several limitations:

- They had to emulate browser behavior, which does not necessarily represent the user experience. Client-side problems are not captured using their method. Further, they couldn't handle connections with the SNI extension, which eliminated 38% of the HTTPS connections that they saw.
- Their observed population isn't representative. All of the monitored users were highly educated and in the U.S., likely using high-end devices. Other populations visit different web pages on different devices.
- Their observed networks aren't representative. They studied well-behaved, well-managed networks with relationships to their research institution. These networks lack the content filters, broken firewalls, and other types of proxies that one might expect to see on messier networks.

Inspired by this research, we performed a similar study from a more advantageous vantage point: a popular web browser. Our data comes from a global population, connecting over many types of networks. The reports include browser data, so that we know exactly what the end user saw in the warning. Some of our findings coincided (e.g., the importance of incomplete chains) but, as expected, we found substantially more problems due to client and network misconfigurations. Additionally, we implemented mitigations for several of the problems that we identified.

11.1.2 Network scans. One way to learn about HTTPS errors is to scan large sets of servers, looking for misconfigurations that cause errors. Holz et al. repeatedly scanned the Alexa Top Million in 2011, finding that 18% of server certificates are expired and about a third are self-signed [18]. In 2013, Durumeric et al. performed 110 Internet-wide scans over fourteen months, reporting that 6% of certificate chains are expired, and 6.4% have missing or wrong intermediates [12].

We find a different ratio of error causes, likely due to our different perspectives. Server scans weigh all servers equally, which is appropriate if one is trying to understand the types of errors that developers make (as these previous studies were). However, we are concerned with the user experience, in which some sites are viewed much more often than others. Further, server scans naturally do not include network and client problems.

11.1.3 Developer motivations. Why do server misconfigurations happen? Given browser warnings, it is surprising that web developers allow server misconfigurations to occur and linger. Fahl et al. surveyed 755 web developers about why they have certificate errors on their websites [14]. A third of developers said they had made a mistake, but two-thirds intentionally deployed non-validating certificates. Their reasons included: testing and development servers don't need HTTPS, the cost of certificates, lack of trust in Certificate Authorities, the URL wasn't meant to be accessed by end users, and the site was no longer operational.

11.2 Studies of TLS proxies

According to two studies, TLS proxies are widespread. Approximately 0.2% of TLS connections to Facebook are transparently proxied [19], and a broader study found that 0.41% of TLS connections in general were proxied [23]. They identified anti-virus software, firewalls, malware, parental controls, and enterprise filters as common types of proxies. We looked for these types of proxies and find that they are also major sources of errors.

Our research question is essentially the complement of these studies. They investigated how often TLS connections are silently intercepted, whereas we aim to identify the causes of user-visible warnings. Their methodologies excluded most TLS connections with warnings because users had to visit the target websites for their analysis code to run. Modern browsers disallow clicking through warnings on `facebook.com` due to HSTS, thereby excluding those connections from the Facebook dataset. The broader study likely included some connections with warnings, but high warning adherence rates (e.g., 70% for Chrome [29]) mean that most would be filtered from their dataset. Further, neither study included websites with server misconfigurations. In contrast, our dataset represents the full spectrum of failed TLS connections.

12 CONCLUSION

In an attack scenario, it is critical that users heed HTTPS certificate error warnings. Large numbers of false alarms make it less likely that they will do so [28, 30]. Spurious warnings also create a poor user experience and hinder HTTPS adoption.

In this paper we have shown that client and network misconfigurations are prominent culprits for spurious certificate warnings. We assigned root causes to certificate reports collected from volunteer Chrome users, and we investigated the small number of root causes – such as incorrect client clocks and insufficient intermediates – which account for vast numbers of warnings. Finally, we proposed, implemented, and evaluated mitigations for the common causes of spurious certificate warnings, replacing about 25% of them in total. Our findings and mitigations are applicable to other browser vendors as well as other types of TLS clients, all of which may be susceptible to client and network misconfigurations that interfere with certificate validations.

ACKNOWLEDGMENTS

We thank Andrew Whalley, Chris Palmer, Emily Schechter, Eric Roman, Lucas Garron, and Noé Lutz for their help with this work.

REFERENCES

[1] [n. d.]. *Captive Portal Interaction (capport).* https://datatracker.ietf.org/wg/capport/about/.

[2] [n. d.]. *Chrome Release Channels.* https://www.chromium.org/getting-involved/dev-channel.

[3] [n. d.]. *Roughtime.* https://roughtime.googlesource.com/roughtime.

[4] 2014. *Comodo EV Chain Issues.* https://community.qualys.com/thread/13775#comment-24990.

[5] 2015. *Kaspersky Lab Forum: clock is being changed by ?virus.* https://forum.kaspersky.com/index.php?showtopic=289198.

[6] 2016. *HTTPS websites fail to load or you receive the error message "Connection is untrusted" when using your web browser with ESET products.* http://support.eset.com/kb3126/?locale=en_US.

[7] 2017. *The Chromium Projects Security FAQ.* https://www.chromium.org/Home/chromium-security/security-faq#TOC-How-does-key-pinning-interact-with-local-proxies-and-filters-.

[8] 2017. *Google Chrome Privacy Whitepaper: Safe Browsing protection.* https://www.google.com/chrome/browser/privacy/whitepaper.html#malware.

[9] Devdatta Akhawe, Bernhard Amann, Matthias Vallentin, and Robin Sommer. 2013. Here's My Cert, So Trust Me, Maybe?: Understanding TLS Errors on the Web. In *Proceedings of the 22Nd International Conference on World Wide Web (WWW '13).* ACM, New York, NY, USA, 59–70. https://doi.org/10.1145/2488388.2488395

[10] Xavier de CarnÃl' de Carnavalet and Mohammad Mannan. 2016. Killed by Proxy: Analyzing Client-end TLS Interception Software. In *NDSS.*

[11] T. Dierks and E. Rescorla. 2008. *The Transport Layer Security (TLS) Protocol Version 1.2.* https://tools.ietf.org/html/rfc5246#section-7.4.2.

[12] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. 2013. Analysis of the HTTPS Certificate Ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC '13).* ACM, New York, NY, USA, 291–304. https://doi.org/10.1145/2504730.2504755

[13] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. 2017. The Security Impact of HTTPS Interception. In *Network and Distributed Systems Symposium (NDSSâĂŽ17).*

[14] Sascha Fahl, Yasemin Acar, Henning Perl, and Matthew Smith. 2014. Why Eve and Mallory (Also) Love Webmasters: A Study on the Root Causes of SSL Misconfigurations. In *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '14).* ACM, New York, NY, USA, 507–512. https://doi.org/10.1145/2590296.2590341

[15] Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. 2015. Improving SSL Warnings: Comprehension and Adherence. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15).* ACM, New York, NY, USA, 2893–2902. https://doi.org/10.1145/2702123.2702442

[16] Adrienne Porter Felt, Robert W. Reeder, Hazim Almuhimedi, and Sunny Consolvo. 2014. Experimenting at Scale with Google Chrome's SSL Warning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14).* ACM, New York, NY, USA, 2667–2670. https://doi.org/10.1145/2556288.2557292

[17] Lucas Garron and David Benjamin. 2015. *An update on SHA-1 certificates in Chrome.* https://security.googleblog.com/2015/12/an-update-on-sha-1-certificates-in.html.

[18] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL Landscape: A Thorough Analysis of the x.509 PKI Using Active and Passive Measurements. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC '11).* ACM, New York, NY, USA, 427–444. https://doi.org/10.1145/2068816.2068856

[19] Lin Shung Huang, Alex Rice, Erling Ellingsen, and Collin Jackson. 2014. Analyzing Forged SSL Certificates in the Wild. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP '14).* IEEE Computer Society, Washington, DC, USA, 83–97. https://doi.org/10.1109/SP.2014.13

[20] Mariko Kobayashi. 2017. *Survey on Behaviors of Captive Portals.* https://www.ietf.org/proceedings/98/slides/slides-98-capport-survey-00.pdf.

[21] Frank Li, Grant Ho, Eric Kuan, Yuan Niu, Lucas Ballard, Kurt Thomas, Elie Bursztein, and Vern Paxson. 2016. Remedying Web Hijacking: Notification Effectiveness and Webmaster Comprehension. In *International World Wide Web Conference.*

[22] Tyler Odean. 2012. *Chromium Blog: Changes to the Field Trials Infrastructure.* https://blog.chromium.org/2012/05/changes-to-field-trials-infrastructure.html.

[23] Mark O'Neill, Scott Ruoti, Kent Seamons, and Daniel Zappala. 2016. TLS Proxies: Friend or Foe?. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16).* ACM, New York, NY, USA, 551–557. https://doi.org/10.1145/2987443.2987488

[24] Tavis Ormandy. 2016. *Kaspersky: SSL interception differentiates certificates with a 32bit hash.* https://bugs.chromium.org/p/project-zero/issues/detail?id=978.

[25] Waseem Patwegar. 2016. *How to Fix Slow or Incorrect Windows Computer Clock.* http://www.techbout.com/fix-slow-incorrect-windows-computer-clock-14287/.

[26] Deborah Salmi. 2015. *Avast Web Shield scans HTTPS sites for malware and threats*. https://blog.avast.com/2015/05/25/explaining-avasts-https-scanning-feature/.

[27] Angela Sasse. 2015. Scaring and Bullying People into Security Won't Work. *IEEE Security and Privacy* (May/June 2015).

[28] David W Stewart and Ingrid M Martin. 1994. Intended and unintended consequences of warning messages: A review and synthesis of empirical research. *Journal of Public Policy & Marketing* (1994), 1–19.

[29] Joel Weinberger and Adrienne Porter Felt. 2016. A Week to Remember: The Impact of Browser Warning Storage Policies. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*. USENIX Association, Denver, CO, 15–25. https://www.usenix.org/conference/soups2016/technical-sessions/presentation/weinberger

[30] M Wogalter. 2006. Purposes and scope of warnings. *Handbook of Warnings (3–9); Wogalter, M., Ed* (2006).