Security Certification in Payment Card Industry: Testbeds, Measurements, and Recommendations

Sazzadur Rahaman[1], Gang Wang[2], Danfeng (Daphne) Yao[1]
[1]Computer Science, Virginia Tech, Blacksburg, VA
[2]Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL
sazzad14@vt.edu,gangw@illinois.edu,danfeng@vt.edu

Abstract

The massive payment card industry (PCI) involves various entities such as merchants, issuer banks, acquirer banks, and card brands. Ensuring security for all entities that process payment card information is a challenging task. The PCI Security Standards Council requires all entities to be compliant with the PCI Data Security Standard (DSS), which specifies a series of security requirements. However, little is known regarding how well PCI DSS is enforced in practice. In this paper, we take a measurement approach to systematically evaluate the PCI DSS certification process for e-commerce websites. We develop an e-commerce web application testbed, BuggyCart, which can flexibly add or remove 35 PCI DSS related vulnerabilities. Then we use the testbed to examine the capability and limitations of PCI scanners and the rigor of the certification process. We find that there is an alarming gap between the security standard and its real-world enforcement. None of the 6 PCI scanners we tested are fully compliant with the PCI scanning guidelines, issuing certificates to merchants that still have major vulnerabilities. To further examine the compliance status of real-world e-commerce websites, we build a new lightweight scanning tool named PciCheckerLite and scan 1,203 e-commerce websites across various business sectors. The results confirm that 86% of the websites have at least one PCI DSS violation that should have disqualified them as non-compliant. Our in-depth accuracy analysis also shows that PciCheckerLite's output is more precise than w3af. We reached out to the PCI Security Council to share our research results to improve the enforcement in practice.

CCS Concepts

• **Security and privacy → Web application security**; *Web protocol security*.

Keywords

Payment Card Industry; Data Security Standard; Internet Measurement; Website Scanning; Data Breach; Web Security; Testbed; E-commerce;

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363195

ACM Reference Format:
Sazzadur Rahaman, Gang Wang, Danfeng (Daphne) Yao. 2019. Security Certification in Payment Card Industry: Testbeds, Measurements, and Recommendations. In *2019 ACM SIGSAC Conference on Computer & Communications Security (CCS'19), November 11–15, 2019, London, UK.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363195

1 Introduction

Payment systems are critical targets that attract financially driven attacks. Major card brands (including Visa, MasterCard, American Express, Discover, and JCB) formed an alliance named Payment Card Industry Security Standards Council (PCI SSC) to standardize the security requirements of the ecosystem at a global scale. The PCI Security Standards Council maintains, updates, and promotes Data Security Standard (DSS) [27] that defines a comprehensive set of security requirements for payment systems. PCI DSS certification has established itself as a global trademark for secure payment systems. According to PCI DSS [27],

*"PCI DSS applies to **all** entities involved in payment card processing – including merchants, processors, acquirers, issuers, and service providers. PCI DSS also applies to **all** other entities that store, process, or transmit cardholder data and/or sensitive authentication data."*

The PCI Security Standards Council plays a major role in evaluating the security and compliance status of the payment card industry participants and supervises a set of entities that are responsible to perform compliance assessments such as Qualified security assessors (QSA) and Approved scanning vendors (ASV). All entities in the PCI ecosystem, including merchants, issuers, and acquirers, need to comply with the standards. PCI standards specify that entities need to obtain their compliance reports from the PCI authorized entities (*e.g.*, QSA and ASV) and periodically submit the reports in order to maintain their status. For example, a merchant needs to submit its compliance report to the acquirer bank to keep its business account active within the bank. Similarly, card issuer and acquirer banks need to submit their compliance reports to the payment brands (*e.g.*, Visa, MasterCard, American Express, Discover, and JCB) to maintain their membership status [27].

However, several recent high-profile data breaches [14, 72] have raised concerns about the security of the payment card ecosystem, specially for e-commerce merchants[1]. A research report from Gemini Advisory [21] shows that 60 million US payment cards have been compromised in 2018 alone. Among the merchants that experienced data breaches, many were known to be compliant with the PCI data security standards (PCI DSS). For example, in 2013, Target leaked 40 million payment card information due to insecure

[1]Merchants that allow *online* payment card transactions for selling products and services are referred to as "e-commerce merchants".

practices within its internal networks [72], despite that Target was marked as PCI DSS compliant. These incidents raise important questions about how PCI DSS is enforced in practice.

In this paper, we ask: *how well are the PCI data security standards enforced in practice? Do real-world e-commerce websites live up to the PCI data security standards?* These questions have not been experimentally addressed before. We first design and develop testbeds and tools to quantitatively measure the degree of PCI DSS compliance of PCI scanners and e-commerce merchants. PCI scanners are commercial security services that perform external security scans on merchants' servers and issue certificates to those who pass the scan. By setting up our testbed, *i.e.*, an e-commerce website with configurable vulnerabilities, we empirically measure the capability of PCI scanners and the rigor of the certification process.

Our results show that the detection capabilities of PCI scanners vary significantly, where even PCI-approved scanners fail to report serious vulnerabilities in the testbed. For 5 of the 6 scanners evaluated, the reports are not compliant with the PCI scanning guidelines [19]. All 6 scanners issued certificates to web servers that still have major vulnerabilities (*e.g.*, sending sensitive information over HTTP). Even if major vulnerabilities are detected (*e.g.*, remotely accessible MySQL), which should warrant an "automatic failure" according to the guideline [19], some PCI scanners still proceed with certification regardless.

Given the weak scanner performance, it is possible that real-world e-commerce websites still have major vulnerabilities. For validation, we build a new lightweight scanning tool and perform empirical measurements on 1,203 real-world e-commerce websites. Note that for independent researchers or third-parties, scanning in the PCI context imposes a new technical challenge, namely the non-intrusive low-interaction constraint. The low interaction constraint, necessary for testing live production sites, makes it difficult to test certain vulnerabilities externally. Traditional penetration testing (pentest) tools are not suitable to test live websites in production environments. For example, pentest tools such as w3af [4] have brute-force based tests which require intense URL fuzzing (*e.g.*, prerequisite for SQL injection, XSS) or sending disruptive payload. The feedback from the PCI Security Council during our disclosure (Section 6) also confirmed this challenge.

Our technical contributions and findings are summarized below.

- We design and develop an e-commerce web application testbed called BuggyCart, where we implant 35 PCI-related vulnerabilities such as server misconfiguration (*e.g.*, SSL/TLS and HTTPS misconfigurations), programming errors (*e.g.*, CSRF, XSS, SQL Injection), and noncompliant practices (*e.g.*, storing plaintext passwords, PAN, and CVV). BuggyCart allows us to flexibly configure vulnerabilities in the testbed for measuring the capabilities and limitations of PCI scanners. We are in the process of open-sourcing BuggyCart and sharing it with the PCI security council (Section 3).
- Using BuggyCart, we evaluated 6 PCI scanning services, ranging from more expensive scanners (*e.g.*, $2,995/Year) to low-end scanners (*e.g.*, $250/Year). The results showed an alarming gap between the specifications of the PCI data security standard and its real-world enforcement. For example, most of the scanners choose to certify websites with serious SSL/TLS and server misconfigurations. None of the PCI-approved scanning vendors detect SQL injection, XSS, and CSRF. 5 out of the 6 scanners are not compliant with the ASV scanning guidelines (Section 4).
- We further evaluated 4 generic web scanners (not designed for PCI DSS), including two commercial scanners and two open-source academic solutions (w3af [4], ZAP [2]). We examine whether they can detect the web-application vulnerabilities missed by PCI scanners. Unfortunately, most of these vulnerabilities still remain undetected. (Section 4).
- We conducted empirical measurements to assess the (in)security of real-world e-commerce websites. We carefully designed and built a lightweight vulnerability scanner called PciCheckerLite. Our solution to addressing the non-intrusiveness challenge is centered at minimizing the number of requests that PciCheckerLite issues per test case, while maximizing the test case coverage. It also involves a collection of lightweight heuristics that merge multiple security tests into one request. Using PciCheckerLite, we evaluated 1,203 e-commerce website across various business categories. We showed that 94% of the websites have at least one PCI DSS violation, and 86% of them contains violations that should have disqualified them as non-compliant (Section 5). Our in-depth accuracy analysis also showed that PciCheckerLite's outputs have fewer false positives than the w3af counterpart (Table 6).

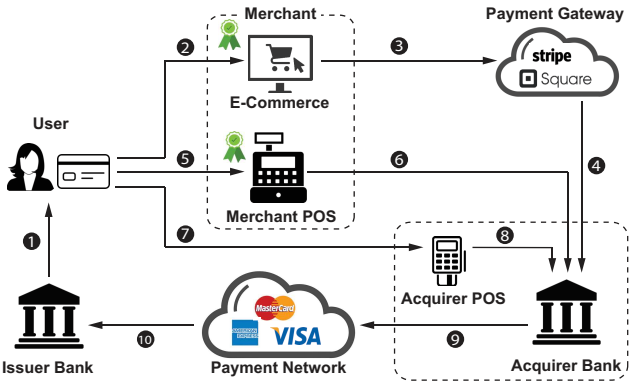

Figure 1: Overview of the payment card ecosystem.

Based on our results, we further discuss how various PCI stakeholders, including the PCI council, scanning providers, banks, and merchants, as well as security researchers, can collectively improve the security of the payment card ecosystem (Section 6).

2 Background on PCI and DSS

We start by describing the background for the security practices, workflow, and standards of the current PCI ecosystem that involves banks, store-front and e-commerce vendors, and software providers. Then, we focus on how merchants obtain security certifications and establish trust with the banks. We discuss how the certification process is regulated and executed.

2.1 Payment Card Ecosystem

The Payment Card Industry (PCI) has established a working system that allows merchants to accept user payment via payment

cards, and complete the transactions with the banks in the backend. Figure 1 shows the relationships between the key players in the ecosystem, including users, merchants, and banks. The user and the merchant may use different banks. The *issuer bank* issues payment cards to the user and manages the user's credit or debit card accounts (step ❶). Users use the payment card at various types of merchants (steps ❷, ❺, and ❼). The *acquirer bank* manages an account for the merchant to receive and route the transaction information (steps ❹, ❻, and ❽). The acquirer bank ensures that funds are deposited into the merchant's account once the transaction is complete via the payment network (steps ❾ and ❿). The payment network, also known as the card brands (*e.g.*, Visa, MasterCard), bridges between the acquirer and the issuer banks.

There are different types of merchants. For merchants that run an e-commerce service (*i.e.*, all transactions are made online), they usually interact with the acquirer bank via a *payment gateway* (*e.g.*, Stripe, Square), which eases the payment processing and integration (❸). For merchants that have a physical storefront, they use point-of-sale (POS) devices, *i.e.*, payment terminals, to collect and transfer user card information to the acquirer bank. They can use either the acquirer bank's POS (❼) or their own POS (❺). The key difference is that acquirer POS directly transfers the card information to the bank without storing the information within the merchant. Merchant POS, however, may store the card information.

Due to the fact that e-commerce websites and merchant POSes need to store card information, the merchants need to prove to the bank that they are qualified to securely handle the information processing. The acquirer bank requires these merchants to obtain PCI security certifications in order to maintain accounts with the bank [11]. Next, we introduce the security certification process.

2.2 PCI Council and Data Security Standard

Payment Card Industry Security Standards Council manages a number of specifications to ensure data security across the extremely complex payment ecosystem. Among all the specifications, only the Data Security Standard (DSS) and Card Production and Provisioning (CPP) are *required*. All the other specifications (shown in Table 8 in the Appendix) are recommended (*i.e.*, optional). CPP is designed to regulate card issuers and manufactures. The Data Security Standard (DSS) is the most important specification that is required to be complied by issuer banks, acquirer banks, and all types of merchants and e-commerce sites, *i.e.*, all systems that process payment cards. Our work is focused on the DSS compliance.

In the PCI Data Security Standard specifications [27], there are 12 requirements that an organization must follow to protect user payment card data. These requirements cover various aspects ranging from network security to data protection policies, vulnerability management, access control, testing, and personnel management. In total, there are 79 more detailed items under the 12 high-level requirements. We summarize them in the Appendix (Table 9).

DSS applies to all players in the ecosystem, including *all merchants and acquirer/issuer banks*. For merchants, they need to approve their compliance to the acquirer bank to open an account for their business. For acquirer and issuer banks, they need to prove their compliance to the card brands (*e.g.*, Visa, MasterCard) for the eligibility of membership.

Table 1: PCI Compliance levels and their evaluation criteria.

Level	Transactions Per Year	Compliance Requirements		
		Self-report with SAQ	Sec Scans by ASV	Sec Audits by QSA
Level 1	Over 6M	Quarterly	Quarterly	Required
Level 2	1M – 6M	Quarterly	Quarterly	Required/Optional
Level 3	20K – 1M	Quarterly	Quarterly	**Not Required**
Level 4	Less than 20K	Quarterly	Quarterly	**Not Required**

We use the merchant as an example to illustrate how DSS compliance is assessed. First, the PCI security standard council provides the specifications and self-assessment questionnaires (SAQ) [11]. Merchants self-assess their DSS compliance and attach the questionnaires in their reports. Second, the merchant must pass the security tests and audits from external entities such as Approved Scanning Vendors (ASV) and the Qualified Security Assessors (QSA). The PCI council approves a list of ASV and QSA [20] for the assessment.

Security scanning is conducted by certified scanners (Approved Scanning Vendors or ASVs) on card processing entities. Security scanning is performed remotely without the need for on-site auditing. Not all the requirements can be automatically verified by the remote scanning (see Table 8 in the Appendix). The PCI council provides an ASV scanning guideline [19], which details the responsibilities of the scanners (see Table 7 in the Appendix).

Self-assessment questionnaires (SAQs) allow an organization to self-evaluate its security compliance [29]. In SAQs, all the questions are close ended. More SAQ analysis is presented in Section 6.

Security audit is carried out by Qualified Security Assessors (QSAs). It requires on-site auditing (*e.g.*, checking network and database configurations, examining software patches, and interviewing employees). As security scanning cannot verify all of the DSS properties, on-site audits are to cover those missing aspects.

Level of compliance varies for different organizations. The compliance level is usually determined by the number of annual financial transactions handled by the organization. Each acquirer bank (or card brand) has its own program for compliance and validation levels. In Table 1, we show the tentative compliance levels that roughly match most of the payment brands [11, 23]. The self-assessment questionnaires (SAQs) and security scanning are required quarterly regardless of the compliance levels. Only large organizations that handle over 1 million transactions a year are required to have the on-site audit (by a QSA). The majority of merchants are small businesses, (*e.g.*, 85% of merchants all over the world have less than 1 million USD web sale [15]). Thus, most online merchants rely on ASV scanners and self-reported questionnaires for compliance assessment.

2.3 Our Threat Model and Method Overview

Threat Model. The certification process is designed as an enforcement mechanism for merchants to hold a high-security standard to protect user data from external adversaries. If the certification process is not well executed, it would allow merchants with security vulnerabilities to store payment card data and interact with banks. In addition, such security certification may also create a false sense of security for merchants. We primarily focus on the automatic server screening by PCI scanners given that all merchants need to pass the scanning. We also briefly analyze the Self-assessment

Questionnaire (SAQs). Our analysis does not cover on-site audits, because *i)* on-site audit is not required for the vast majority of the merchants and *ii)* it is impossible to conduct analysis experiments on on-site audits without partnerships with service providers.

Methodology Overview. To systematically measure and compare the rigor of the compliance assessment process, our methodology is to build a semi-functional e-commerce website as a testbed and order commercial PCI scanning services to screen and certify the website. The testbed allows us to easily configure website instances by adding or removing key security vulnerabilities that PCI DSS specifies. We leverage this testbed to perform controlled measurements on the certification process of a number of PCI scanners. In addition to the controlled experiments, we also empirically measure the security compliance of real-world e-commerce websites with a focus on a selected set of DSS requirements. In the following, we describe our detailed measurement methodologies and findings.

3 Measurement Methodology

In this section, we describe our measurement methodology for understanding how PCI scanners perform data security standard (DSS) compliance assessment and issue certificates to merchants. The core idea is to build a re-configurable testbed where we can add or remove key security vulnerabilities related to DSS and generate testing cases. By ordering PCI scanning services to scan the testbed, we collect incoming network traffic as well as the security compliance reports from the scanning vendors. In the following, we first describe the list of vulnerabilities that our testbed covers, and then introduce the key steps to set up the e-commerce frontend.

3.1 Security Test Cases

The testbed contains a total of 35 test cases, where each test case represents a type of security vulnerabilities. Running a PCI scanner to scan the testbed could reveal vulnerabilities that the scanner can detect, as well as those that the scanner fails to report. We categorize the 35 security test cases i_1–i_{35} into four categories, namely *network security*, *system security*, *application security*, and *secure storage*. Note that there are 29 test cases in the first three categories are within the scope of ASV scanners (*i.e.*, ASV testable cases). The other 6 cases under "secure storage" cannot be remotely verified. We include these cases to illustrate the limits of ASV scanners.

(1) **Network security (14 test cases).** These testing cases are related to network security properties, including firewall status, (i_1), the access to critical software from network (i_2–i_4), default passwords (i_5–i_6), the usage of HTTP to transmit sensitive data (*e.g.*, customer or admin login information) (i_7), and SSL/TLS misconfigurations (i_{12}–i_{18}).

(2) **System security (7 test cases).** These test cases are related to system vulnerabilities, including vulnerable software (i_{19}–i_{20}), server misconfigurations (i_{29}–i_{32}), and HTTP security headers (i_{33}).

(3) **Web Application security (8 test cases).** These test cases are related to application-level problems including SQL injections (i_{21}–i_{22}), not following secure password guidelines (i_{23}–i_{24}), the integrity of Javascripts from external sources (i_{25}), revealing crash reports (i_{26}), XSS (i_{27}) and CSRF (i_{28}).

(4) **Secure storage (6 test cases).** Secure storage is impossible to verify through external scans. Thus, DSS does not require PCI scanners to test these properties, such as storing sensitive user information (i_8), storing and showing PAN in plaintext (i_9–i_{11}), and insecure ways of storing passwords (i_{34}–i_{35}). In PCI DSS, merchants need to fill out the self-assessment questionnaire about how they handle sensitive data internally. We choose to include these vulnerabilities in the testbed for highlighting the fundamental limitations of external scans on some important aspects of server security.

Must-fix Vulnerabilities. These test cases are designed following the official ASV scanning guideline [19] and the PCI data security standard (DSS) [27]. Among the 35 cases, 29 are within the scope (responsibility) of ASV scanners that can be remotely tested. After vulnerabilities are detected, website owners are required to fix any vulnerabilities that have a CVSS score ≥ 4.0, and any vulnerabilities that are marked as mandatory in PCI DSS. CVSS (Common Vulnerability Scoring System) measures the severity of a vulnerability (score 0 to 10). The CVSS scores in Table 3 are calculated using CVSSv3.0 calculator [1]. Vulnerabilities that have no CVSS score are marked as "N/A". If the website owner fails to resolve the "must-fix" vulnerabilities, a scanner should not issue the compliance certification. As shown in Table 3, 26 out of the 29 testable cases are required to be fixed. Three cases (vulnerability 3, 4, and 18) are not mandatory to fix. For example, exposing OpenSSH to the Internet (case-3) does not mean immediate danger as long as the access is well protected by strong passwords or SSH keys.

Completeness and Excluded Cases. When building our BuggyCart testbed and the PciCheckerLite prototype, we exclude five mandatory ASV scanning cases: *i)* backdoors or malware, *ii)* DNS server vulnerabilities, *e.g.*, unrestricted DNS zone transfer, *iii)* vulnerabilities in mail servers, *iv)* vulnerabilities in hypervisor and virtualization components, and *v)* vulnerabilities in wireless access points. Most of them (namely, *ii*, *iii*, *iv*, and *v*) are not relevant, as they involve servers or devices outside our testbed or an application server. In the first category, it is difficult to design a generic network-based testing case. We also exclude the non-mandatory cases (shown in the last 4 rows of Table 7 in the Appendix).

Note that ASV testable cases only represent a subset of PCI DSS specifications [27] because some specifications are not remotely verifiable. There are specifications related to organization policies, which are impossible to verify externally, *e.g.*, "restricting physical access to cardholder data" (DSS req. 9), and "documenting the key management process" (DSS req. 3.6). They can only be assessed by onsite audits, which unfortunately are not applicable to the majority of e-commerce websites and small businesses (see Table 1). We will discuss this further in Section 6.

Our PciCheckerLite prototype in Section 5 scans 17 test cases in Table 5, which are a subset of the 29 externally scannable rules in Table 3. When scanning live production websites, we have to eliminate cases that require intrusive operations such as web crawling, URL fuzzing, or port scanning.

3.2 Testbed Architecture and Implementations

A key challenge of measuring PCI scanners is to interact with PCI scanners like a real e-commerce website does, in order to obtain reliable results. This requires the testbed to incorporate most (if not

all) of the e-commerce functionality to interact with PCI scanners and reflect the scanners' true performance. For this reason, we choose the OpenCart [26] as the base to build our testbed. OpenCart is a popular open source PHP-based e-commerce solution used by real-world merchants to build their websites. This allows us to interact with PCI scanners in a realistic manner to ensure the validity of measurement results.

Testbed Frontend. The frontend of our testbed supports core e-commerce functionality, such as account registration, shopping cart management, and checkout and making payment with credit cards. The code of the website[2] is based on OpenCart. We rewrote the OpenCart system by integrating all 35 security vulnerabilities and testing cases. We deployed the website using Apache HTTP server and MySQL database. Our testbed automatically spawns a website instance following a pre-defined configuration. We used OpenSSH as the remote access software and Phpmyadmin to remotely manage the MySQL database. We hosted our website in Amazon AWS in a single *t2.medium* server instance with Ubuntu 16.04. We obtained a valid SSL certificate to enable HTTPS from *Let's Encrypt* [25].

We set up the website solely for research experiment purposes. Thus, it does not have a real payment gateway. Instead, we set up a dummy payment gateway that imitates the real gateway Cardconnect [22]. The website forwarded credit card transactions to this dummy payment gateway. The dummy endpoint for Cardconnect is implemented using *flask-restful* framework. We modified the `/etc/hosts` file of our web server to redirect the request. During our experiments, our server did not receive any real payment transaction requests. We further discuss research ethics in Section 3.3.

Implementing Security Test Cases. Next, we describe the implementation details of the 35 security test cases in Table 3.

For the network security category, we implement test cases i_1 to i_3 by changing inbound traffic configurations within the Amazon AWS security group. Test case i_4 (administer access over Internet) is implemented by changing *phpmyadmin* configuration. For test case i_5 (default SQL password), we do not set any password for "*root*" and enable access from *any remote host*. Test case i_5 is implemented by configuring phpmyadmin (no password for user "*root*"). Test case i_7 is set to keep port 80 (HTTP) open without a redirection to port 443 (HTTPS). Test cases i_{12}, i_{14}, i_{16}, and i_{17} are implemented by using default certificates from Apache. Test cases i_{13} and i_{18} are implemented by changing *SSLCipherSuite* and *SSLProtocol* of the Apache server. For test case i_{15}, we configure the Apache server to use a valid certificate but with a wrong domain name.

For the system security category, we implement test cases i_{19}–i_{20} by installing software that are known to be vulnerable. For test case i_{19}, we use *OpenSSL 7.2*, which is vulnerable to privilege escalation and timing side channel attacks. For test case i_{20}, we used *phpmyadmin 4.8.2* which is known to be vulnerable to XSS. We implemented test cases i_{29} to i_{33} by changing the configurations of the Apache server. For test case i_{33} (HTTP security header) in particular, we consider X-Frame-Options, X-XSS-Protection, X-Content-Type-Options, and Strict-Transport-Security.

For the web application security category, we implement test cases i_{21} to i_{28} by modifying OpenCart source code [26]. Regarding secure password guidelines, we disable password retry restrictions for both users and administrators (test case i_{23}), disable the length checking of passwords (test case i_{24}). For SQL injection, we modify the admin login (test case i_{21}) and customer login (test case i_{22}) code to implement SQL injection vulnerabilities. For admin login, we simply concatenate user inputs without sanitation for the login query. For the customer login, we leave an SQL injection vulnerability at the login form. Given that the user password is stored as unsalted MD5 hashes, we run the login query by concatenating the MD5 hash of the user-provided password, which is known to be vulnerable to SQL injection [5]. For XSS and CSRF, we implant an XSS vulnerability in the page of editing customer profiles, by allowing HTML content in the "first name" field (test case i_{27}). By default, Opencart does not have any protection against CSRF (test case i_{28}). For test case i_{26} (displaying errors), we configure OpenCart to reveal crash reports (an insecure practice, which gives away sensitive information). Opencart by default does not check the integrity of Javascript code loaded from external sources (test case i_{25}).

For the secure storage category, we modify the Cardconnect extension to store CVV in our database (test case i_8) and the full PAN (instead of the last 4 digits) in the database in plaintext (test case i_{10}). We add an option to encrypt PANs before storing, but the encryption key is hardcoded (test case i_{11}). We also update the customers' order history page to show the unmasked PAN for each transaction (test case i_9). Finally, the testbed stores the raw unsalted MD5 hash of passwords for customers (test case i_{34}) and plaintext passwords for admins (test case i_{35}).

3.3 Research Ethics

We have taken active steps to ensure research ethics for our measurement on PCI scanners (Section 4). Given that our testbed is hosted on the public Internet, we aim to prevent real users from accidentally visiting the website (or even putting down credit card information). First, we only put the website online shortly before the scanning experiment. After each scanning, we immediately take down the website. Second, the website domain name is freshly registered. We never advertise the website (other than giving the address to the scanners). Third, we closely monitor the HTTP log of the server. Any requests (*e.g.*, for account registration or payment) that are not originated from the scanners are dropped. Network traffic from PCI scanners are easy to distinguish (based on IP and User-Agent) from real user visits. We did not observe any real user requests or payment transactions during our experiments.

All PCI scanners run automatically without any human involvement from the companies. We order and use the scanning services just like regular customers. We never actively generate traffic to the scanning service, and thus our experiments do not cause any interruptions. Our experiments follow the terms and conditions specified by the scanning vendors, which we carefully examined. We choose to anonymize the PCI scanners' names since some scanning vendors strictly forbid publishing any benchmark results. We argue that publishing our work with anonymized scanner names is sufficient for the purpose of describing the current security practice in the payment card industry, as the security issues reported are likely industry-wide, not unique to the individual scanners evaluated. In addition, anonymization would help alleviate the bias toward individual scanners and potential legal issues [54].

[2]The URL was `www.rwycart.com`. We took the site offline after the experiment.

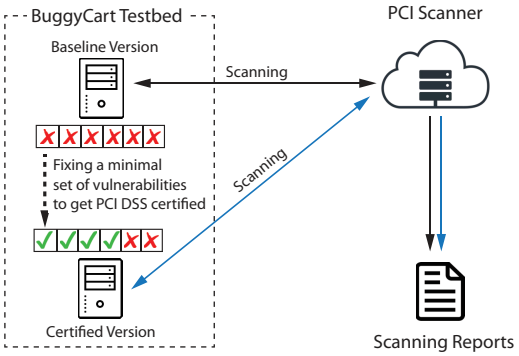

Figure 2: Illustration of the baseline scanning and the certified version. A PCI scanner iteratively scans the testbed. The initial scan (baseline) is on the original testbed with all 35 vulnerabilities. The certified version is the testbed version where the testbed successfully passes the scanning after we iteratively fix a minimal set of vulnerabilities in the testbed. In Table 3, we report the scanning results on both versions of the testbed for each scanner.

In Section 5, we also carefully design our experiments when evaluating the compliance of 1,203 websites. The experiment is designed in a way that generates minimal footprints and impact on the servers, in terms of the number of connection requests to the servers. Our client is comparable to a normal client and does not cause any disruption to the servers. For example, we quickly closed the connection, after finding out whether or not an important port is open. More details are be presented in Section 5.

4 Evaluation of PCI Scanners

Our first set of experiments is focused on evaluating PCI scanners to answer the following research questions. Later in Section 5, we will introduce our second set of experiments on measuring the security compliance of real-world e-commerce websites.

- How do various PCI scanners compare in terms of their detection capabilities? (Section 4.1)
- What are the security consequences of inadequate scanning and premature certification? (Section 4.2)
- How are web scanners (commercial or open-source ones) compared with PCI scanners in terms of detection capabilities? (Section 4.3)

We selected 8 U.S. based PCI DSS scanners as shown in Table 2. The selection process is as follows. From the list of approved vendors [20][3], we found 85 of them operate globally. Out of these 85, we aimed to identify a set of ASVs that appear to be of high quality (*e.g.*, judging from the company's reputations and websites) and somewhat affordable (due to our limited funding), while also covering different price ranges. We identified 6 such scanners. For 3 of them, the prices are publicly available. For the other 3 scanners, we emailed them through our rwycart.com email addresses. 2 of them (Scanner7 and Scanner8) did not provide their price quotations, which forced us to drop them from our evaluation (due to our organization policies). During our search, we also found that some website owners used non-ASV scanners. Thus, we also included 2 non-ASVs that have good self-reported quality. Non-approved scanners offer commercial PCI scanning services, but are not on the ASV list [20] of the PCI council[4]. Because of the legal constraints imposed by the terms and conditions of scanners, we cannot reveal scanners' names. Researchers who wish to reproduce or extend our work for scientific purposes without publishing scanner names are welcome to contact the authors.

Table 2: Prices of PCI scanners and the actual costs.

PCI Scanners	Price	Spent Amount	PCI SSC Approved?
Scanner1	$2,995/Year	$0 (Trial)	Yes
Scanner2	$2,190/Year	$0 (Trial)	Yes
Scanner3	$67/Month	$335	No
Scanner4	$495/Year	$495	Yes
Scanner5	$250/Year	$250	Yes
Scanner6	$59/Quarter	$118	No
Scanner7	Unknown	N/A	Yes
Scanner8	$350/Year	N/A	Yes
Total	-	$1198	-

We conducted experiments successfully with 6 of the scanners (without Scanner7 and Scanner8 for the reason mentioned above). We use the email address (`wayne@rwycart.com`) associated with the testbed e-commerce website to register accounts at the scanning vendors. Table 2 shows the prices of these 6 vendors. For Scanner2 and Scanner1, we completed our experiments within the trial period (60 days for Scanner2 and 30 days for Scanner1). The trial-version and the paid-version offer the same features and services.

Iterative Test Design. Given a PCI scanner, we carry out the evaluation in two high-level steps shown in Figure 2. Every scanner first runs on the same baseline testbed with all the vulnerabilities built in. Then we remove a *minimal* set of vulnerabilities to get the testbed certified for PCI DSS compliance. The final certified instance of the testbed may be different for different scanners, as high-quality scanners require more vulnerabilities to be fixed, having fewer remaining (undetected) vulnerabilities on the testbed.

(1) **Baseline Test.** We spawn a website instance where all 35 vulnerabilities are enabled (29 of them are remotely verifiable). Then we order a PCI scanning service for this testbed. During the scanning, we monitor the incoming network traffic. We obtain the security report from the scanner, once the scanning is complete.

(2) **Certified Instance Test.** After the baseline scanning, we modify the web server instance according to the obtained report. We perform all the fixes required by the PCI scanner and order another round of scanning. The purpose of this round of scanning is to identify the *minimal* set of vulnerabilities that need to be fixed in order to pass the PCI DSS compliance certification.

In summary, we perform the following steps for each scanner: *i)* implant vulnerabilities under each test case in the testbed, *ii)* run

[3] As of April 30, 2019, 97 companies are approved by the PCI Council as the approved scanning vendors (ASVs) [20].

[4] To become an ASV, a scanner service needs to pay a fee and go through a testbed-based approval evaluation supervised by the PCI Council.

Table 3: Testbed scanning results. "Baseline" indicates the scanning results on our testbed when all the 35 vulnerabilities are active. "Certified" indicates the scanning results after fixing the minimum number of vulnerabilities in order to be compliant. "○", "◐", "●" means severity level of low, medium, and high respectively according to the scanners. "✗" mean "undetected", "✓" means "fixed in the compliant version", "✓*" means "fixed as a side-effect of another case". The "website scanners" represent a separate experiment to determine whether website scanners can help to improve coverage. We ran the website scanners on test cases that were not detected by the PCI ASV scanners. "N/A" means "not testable by an external scanner". "-" means "testable but do not need to tested". The "Must Fix" column shows the vulnerabilities that must be fixed by the e-commerce websites in order to be certified as PCI DSS compliant.

Rq.	Test Cases	Vul. Location	In ASV Scope?	CVSS Score	Must Fix?	Scanner2		Scanner5		Scanner4 / Scanner1		Scanner6 (not aprvd.)		Scanner3 (not aprvd.)		Website Scanners			
						Baseline	Certified	Baseline	Certified	Baseline	Certified	Baseline	Certified	Baseline	Certified	Scanner2W	Scanner5W	W3af	ZAP
1.1	1. Firewall detection	OS	Y	N/A	Y	○	○	○	○	○	○	✗	✗	✗	✗	-	-	-	-
1.2	2. Mysql port (3306) detection	OS	Y	N/A	Y	●	✓	◐	✓	○	○	○	○	○	○	-	-	-	-
	3. OpenSSH detected	OS	Y	N/A	N	◐	✓	○	○	○	○	○	○	○	○	-	-	-	-
	4. Remote access to Phpmyadmin	Apache	Y	N/A	N	◐	✓	○	○	○	○	○	○	○	○	-	-	-	-
2.1	5. Default Mysql user/password	Mysql	Y	8.8	Y	●	✓	○	✓*	●	✓	●	✓	●	✓	-	-	-	-
	6. Default Phpmyadmin passwords	Apache	Y	8.8	Y	✗	✓*	✗	✗	✗	✗	○	○	✗	✗	-	-	-	-
2.3	7. Sensitive information over HTTP	Apache	Y	8.1	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
3.2	8. Store CVV in DB	Webapp	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
3.3	9. Show unmasked PAN	Webapp	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
3.4	10. Store plaintext PAN	Webapp	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
3.5	11. Hardcoded key for encrypting PAN	Webapp	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
4.1	12. Use untrusted selfsigned cert.	Apache	Y	9.8	Y	◐	✓	◐	✓	◐	✓	◐	✓	✗	✗	-	-	-	-
	13. Insecure block cipher (Sweet32)	Apache	Y	7.5	Y	◐	✓	◐	✓	○	○	◐	✓	✗	✗	-	-	-	-
	14. Expired SSL cert.	Apache	Y	5.3	Y	◐	✓	◐	✓	●	✓	◐	✓	✗	✗	-	-	-	-
	15. Wrong domain names in cert.	Apache	Y	5.3	Y	◐	✓	✗	✗	○	○	○	○	✗	✗	-	-	-	-
	16. DH modulus <= 1024 Bits	Apache	Y	5.3	Y	○	✓*	◐	✓	◐	✓	◐	✓	✗	✗	-	-	-	-
	17. Weak Hashing in SSL cert.	Apache	Y	5.3	Y	◐	✓	✗	✓*	✗	✓*	◐	✓	✗	✗	-	-	-	-
	18. TLS 1.0 supported	Apache	Y	3.7	N	●	✓	●	✓	◐	✓	✗	✗	✗	✗	-	-	-	-
6.1	19. Vulnerable OpenSSH (7.2)	OS	Y	7.8	Y	●	✓	●	✓	●	✓	✗	✗	✗	✗	-	-	-	-
	20. Vulnerable Phpmyadmin (4.8.3)	Apache	Y	6.5	Y	◐	✓	✗	✗	✗	✗	✗	✗	✗	✗	-	-	-	-
6.5	21. Sql inject in admin login	Webapp	Y	9.8	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	22. Sql inject in customer login	Webapp	Y	9.8	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
	23. Disable password retry limit	Webapp	Y	5.3	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	24. Allow passwords with len <8	Webapp	Y	5.3	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	25. Javascript source integrity check	Webapp	Y	9.8	Y	●	✓	✗	✗	✗	✗	✗	✗	✗	✗	-	-	-	-
	26. Don't hide program crashes	Webapp	Y	6.5	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	27. Implant XSS	Webapp	Y	6.1	Y	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
	28. Implant CSRF	Webapp	Y	8.8	Y	◐	✓	✗	✗	✗	✗	✗	✗	✗	✗	-	-	-	-
6.6	29. Extraction of server info.	Apache	Y	5.3	Y	◐	✓	○	○	○	○	○	○	○	○	-	-	-	-
	30. Browsable web directories	Apache	Y	7.5	Y	◐	✓	◐	✓	◐	✓	◐	✓	◐	✓	-	-	-	-
	31. HTTP TRACE/TRACK enabled	Apache	Y	4.3	Y	◐	✓	◐	✓	●	✓	◐	✓	●	✓	-	-	-	-
	32. phpinfo() statement is enabled	Apache	Y	5.3	Y	◐	✓	◐	✓	○	○	●	✓	✗	✗	-	-	-	-
	33. Missing security headers in HTTP	Apache	Y	6.1	Y	◐	✓	◐	✓	◐	✓	○	○	✗	✗	-	-	-	-
8.4	34. Store unsalted customer passwords	Webapp	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	35. Store plaintext passwords	Webapp	N	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Baseline: #Vul. Detected (Total detectable: 29)						**21**	-	**16**	-	**17**	-	**16**	-	**7**	-	-	-	-	-
Certified: #Vul. Remaining (#Vul. detected, but no need to fix)						-	**7(0)**	-	**15(3)**	-	**18(7)**	-	**20(7)**	-	**25(4)**	-	-	-	-

the PCI scanning, *iii)* fix all the vulnerabilities that the scanner mandates to fix in order to be PCI DSS compliant, *iv)* run the scanning again, and *v)* record the certified version of the testbed.

4.1 Comparison of Scanner Performance

We found that the security scanning capabilities vary significantly across scanners, in terms of *i)* the vulnerabilities they can detect and *ii)* the vulnerabilities they require one to fix in order to pass the certification process. Once passed, the website becomes PCI DSS compliant. The experimental results are presented in Table 3.

Scanner2. Scanner2 is the most effective PCI scanner in our evaluation, and successfully detected 21 out of the 29 externally detectable cases. The most important case that Scanner2 missed is the use of HTTP protocol to transmit sensitive information (test case 7). We fixed *21* vulnerabilities in our testbed to become PCI compliant in Scanner2. Most of the fixes are intuitive, except fixing Javascript

source integrity checking (Case 25) and CSRF (Case 28). We added Javascript integrity checking for scripts that are loaded from external sources (Case 25). We used a dynamic instrumentation based plugin to protect OpenCart against CSRF attacks (Case 28). This plugin instruments code for generating and checking of CSRF tokens in OpenCart forms. Sometimes, fixing one vulnerability effectively eliminates another vulnerability that Scanner2 fails to detect. For example, Scanner2 did not detect default usernames and passwords for *Phpmyadmin* (Case 6); however, this vulnerability no longer exists after we disable the remote access to *Phpmyadmin* (Case 4).

Scanner5. In the baseline test (*i.e.*, when all the vulnerabilities were in place), Scanner5 detected 16 out of the 29 cases. To obtain a Scanner5 compliant version, we had to fix 13 vulnerabilities. Two of the vulnerabilities (Test cases 5 and 17) are fixed as a side effect of fixing other vulnerabilities (Test cases 2 and 12). Scanner5 failed to report the use of a certificate with the wrong hostname, which is a serious vulnerability exploitable by hackers to launch man-in-the-middle attacks. Scanner5 did not report the use of HTTP to transmit sensitive information (*i.e.*, login and register forms in *rwycart*). Interestingly, Scanner5 detected the use of HTTP to log on to *PhpMyAdmin*. In addition, Scanner5 did not report the use of scripts from external sources (Case 25).

Scanner1 and Scanner4. Scanner4 uses Scanner1's scanning infrastructure for ASV scanning. So we present the experimental results of both scanners under the same column. Scanner1 detects 17 vulnerabilities. However, it only requires fixing 10 of them to be PCI DSS compliant. Some of the high and medium severity vulnerabilities are not required to fix, including remotely accessible Mysql (Case 2), certificates with wrong hostnames (Case 15), and missing security headers (Case 33). The vulnerability of weak hashing in SSL/TLS certificates (Case 17) was fixed as a side effect of using a real certificate from Let's Encrypt (Case 12).

Scanner6 and Scanner3. Scanner6 and Scanner3 are not on the approved scanning vendors (ASVs) list [20] provided by the PCI council. Compared with other approved scanners, they detected a fewer number of vulnerabilities. Scanner6 detected 16 vulnerabilities, whereas Scanner3 detected 7. We fixed 9 of the vulnerabilities for Scanner6 and 3 for Scanner3 in order to be compliant. Both Scanner6 and Scanner3 detected remotely accessible Mysql (Case 2), but do not require us to fix them. Scanner3 missed all the SSL/TLS and certificate related vulnerabilities (Test cases 12-18), while Scanner6 detected most of them. However, Scanner6 did not require us to fix certificates with wrong hostnames (Test case 15). We cannot conclude that unapproved scanners perform worse than approved scanners, due to the small sample size.

A Case Study of False Positives. During our experiment, we find Scanner2 produced a false positive under the SQL injection test. Scanner2 recently incorporated an experimental module to find blind SQL detection, by sending specially crafted parameters to the web server. If the server returns different responses, then it determines that the server has accepted and processed the parameter (*a.k.a* vulnerable). However, this detection procedure fails on a common e-commerce scenario: supporting multiple currencies. OpenCart allows users to select the currency for a product. If a currency is clicked, it updates the currency of the current page. The server records all the parameters of the current page under a hidden field so that it can recreate the page later (Listing 1). Note that Scanner2's specially-crafted parameters are also recorded, which makes Scanner2 believe that there exists a difference in the output under different values of the parameter, which is actually a false positive. Nevertheless, we fixed it by changing the BuggyCart code to be certified by Scanner2.

Listing 1: The difference in the output after injecting a parameter named *name* with an empty value "" vs. "yy".

```
<input type="hidden" name="redirect"
value="http://www.rwycart.com/upload
/index.php?...&amp;product_id=49&amp;name="/>

-------- vs --------

<input type="hidden" name="redirect"
value="http://www.rwycart.com/upload
/index.php?...&amp;product_id=49&amp;name=yy"/>
```

Network Traffic Analysis. We collected the incoming network requests from each of the scanners using the access log of our testbed. During the baseline experiment, Scanner5, Scanner6 and Scanner3 sent 23,912, 39,836, and 31,583 requests, respectively and finished within an hour. Scanner4 and Scanner1 sent 147,038 requests and took more than 3 hours to finish. Scanner2 sent 64,033 requests within 2.5 hours. The reason why we received such a high traffic volume is that the PCI scanners were attempting to detect vulnerabilities such as XSS, SQL injection that require intensive URL fuzzing, crawling and parameter manipulations. This confirms that the PCI scanners have at least attempted to detect such vulnerabilities but were just unsuccessful.

4.2 Impacts of Premature Certification

Some scanners choose to simply report vulnerabilities without marking the e-commerce website as non-compliant. Below, we discuss the security consequences of premature certifications. Some of the incomplete scanning and premature certification issues can be prevented, if the scanners follow the ASV guidelines [19].

Network Security Threats. According to the ASV scanning guideline, SSL/TLS vulnerabilities (Test cases 12–17) should lead to automatic failure of certification, which is clearly necessary due to the man-in-the-middle threats. Only Scanner2 detected all these cases. Scanner3 does not detect any of these SSL/TLS vulnerabilities. In addition, a website should be marked as non-compliant if sensitive information is communicated over HTTP (Test case 7). However, none of the ASV scanners detected this issue in our testbed. This vulnerability can be avoided by configuring the server to automatically redirect all the HTTP traffic to HTTPS. Because none of the 6 scanners detected this vulnerability, it is likely that this HTTP issue exists on real-world e-commerce websites. Our later evaluation on 1,203 websites that process online payment shows 169 of them do not redirect their HTTP traffic to HTTPS (Section 5).

Our Test case 2 embeds a database access vulnerability, allowing the database to be accessible from the Internet. All the scanners detected this vulnerability. However, only Scanner2 and Scanner5 mark this issue as an automatic failure (*i.e.*, non-compliant). The other scanners report it as "low/information", not as a required fix, despite the ASV scanning guideline [19] recommends that to be marked as non-compliant. Our evaluation later on websites that accept payment card transactions shows that 59 out of 1,203 websites opened the Mysql port (3306) to the Internet (Section 5).

System Security Threats. The ASV scanning guideline [19] suggests to test and report vulnerable remote access software. 4 out of the 6 scanners detected vulnerable OpenSSH software (Test case 19). Under Test case 20, only Scanner2 detected vulnerable *phpmyadmin*, while others failed. Although all scanners noticed the Test case 29 (extracted server information), only Scanner2 required a fix for compliance. The ASV scanning guideline [19] also recommends reporting automatic failure if browsable web directories are found (Test case 30). All scanners detected this vulnerability. Scanner6 detected missing security headers (Test case 33), but did not ask us to fix it, while Scanner3 failed to detect it.

Web Application Threats. The scanners' performance is particularly weak for this category. Out of the 8 test cases, only 2 were detected by Scanner2 (tampered Javascript and CSRF). None of the cases was detected by other PCI scanners.

4.3 Evaluation of Website Scanners

The above results suggest that some web application vulnerabilities are difficult to detect. The follow-up question is, *can specialized website scanners detect these vulnerabilities?* To answer this question, we ran four website scanners on our BuggyCart testbed, including two commercial ones (from Scanner2 and Scanner5) and two open source scanners (w3af [4] and ZAP [2]). w3af and ZAP are state-of-the-art open source web scanners, are actively being maintained and are often used in academic research [49, 50, 67]. The two commercial web scanners are from reputable companies. Scanner2W and Scanner2 are from the same company, where the website scanner is marketed as a different product from PCI scanner. It is the same for Scanner5W and Scanner5.

We conducted the baseline test for the four website scanners. Note that these web scanners do not produce certificates. The results are shown in the last four columns of Table 3. Since they are website scanners, we only expect them to cover web application vulnerabilities (Test case 7, 21, 22, 23, 24, 26, 27). Unfortunately, none of the commercial scanners detect these web application vulnerabilities. W3af reported the use of HTTP protocol to communicate sensitive information (case 7), but missed other web application vulnerabilities. ZAP detected the SQL injection vulnerability in the customer login page (case 22), but missed the SQL injection vulnerability in the admin login page (case 21). Noticeably, ZAP also missed the XSS vulnerability we implanted (case 27).

Summary of Testbed Findings. The detection capabilities vary significantly across scanners. Our experiments show that 5 out of 6 PCI scanners are not compliant with the ASV scanning guidelines [19] by ignoring detected vulnerabilities and not making them "automatic failures". Most of the common web application vulnerabilities (*e.g.*, SQL injection, XSS, CSRF) are not detected by the 6 PCI scanners (only Scanner2 detected CSRF), despite the requirements of the PCI guidelines. Out of the 4 website scanners, only ZAP detected one of the two SQL injection cases.

Admittedly, black-box detection of vulnerabilities such as XSS and SQL injection is difficult. Typical reasons for missed detection are *i)* failure to locate the page due to incomplete discovery and/or *ii)* that detection heuristics are limited or easily bypassed. In our testbed, SQL injection vulnerabilities (21, 22) are placed in the login pages. CSRF vulnerabilities are present in all forms. The scanners we tested used web crawling with URL fuzzing to detect hidden pages, URLs, and functions. Often, we are unable to pinpoint the exact reasons why the tools fail in these cases. Novel detection techniques, such as guided fuzzing [49] and taint tracking [73], have been proposed by the research community. Future work is needed to evaluate their applicability in the specific PCI context.

5 Measurement of Compliant Websites

The alarming security deficiencies in how PCI scanners conduct the compliance certification motivate us to ask the following questions: *How secure are e-commerce websites? What are the main measurable vulnerabilities in e-commerce websites?* As such we designed another set of real-world experiments where we aim to measure the security of e-commerce websites with respect to the PCI DSS guideline. To do so, we need to address several technical questions.

What Tools to Use? The key enabler of this measurement is a new tool we developed named PciCheckerLite. We use basic Linux tools (*e.g.*, *nc*, *openssl*) and Java net URL APIs to implement the system. Below, we focus on the key design concepts of PciCheckerLite in order to work with *live websites*.

What Security Properties to Check? A key requirement of this experiment to make sure that we do not disrupt or negatively impact websites being tested. Out of the 29 externally verifiable cases in Table 3, we choose a subset of 17 cases for this experiment, as shown in Table 5. The sole reason of selecting these cases for PciCheckerLite is that we could implement these tests in a non-intrusive manner, leaving a minimum footprint, *i.e.*, having a minimum impact on the servers. We categorize these cases to high, medium and low severity based on *i)* the attacker's gain and *ii)* the attack difficulty. Cases that are immediately exploitable by any arbitrary attacker to cause large damages are highly severe, for example, the use of default passwords (Test case 5), insecure communications (Test cases 7, 12, 13, 16, 17), vulnerable remote access software (Test case 19), browsable web directories (Test case 22), and supporting HTTP TRACE method (Test case 23). Cases that substantially benefit any arbitrary attacker but require some efforts to exploit are marked as medium severity, *e.g.*, test cases 2, 14, 15, 25, 29, and 33. For example, scripts loaded from external sources can steal payment card data (Test case 25), but attackers need to craft the malicious scripts [32]. Low-risk issues are marked as low severity (Test case 3, 18). The categories are consistent with Table 3 as high and medium severity cases correspond to "must-fix" vulnerabilities. The two low-severity cases are not required to be fixed to be PCI-compliant.

Implementing PciCheckerLite**.** Our goal is to minimize the number of requests that PciCheckerLite issues per test case, while maximizing the test case coverage. It involves a collection of lightweight heuristics that merge multiple tests into a single request. For example, for most of the HTTP-related tests, we reuse a single response from the server. Test cases 25, 29, and 33 are covered and resolved by one single HTTP request to retrieve the main page and analyzing the response header. Test cases 12–18 are covered by one certificate fetching. For case 30 (browsable directories enabled) PciCheckerLite conducts a code-guided probe and avoids crawling web pages. It discovers file paths in the code of the landing page and then probes the server with requests for accessing path prefixes. The implementation details are given in the Appendix.

Table 4: Number of e-commerce websites that have at least one vulnerability and those that have at least one "must-fix" vulnerability. In total, 1,203 sites are tested including 810 sites chosen from different web categories, and 393 sites chosen from different Alexa ranking ranges.

E-commerce Websites		# of Vulnerable Websites	
		Must-fix Vul.	All Vul.
Category (810)	Business (122)	106	113
	Shopping (163)	135	143
	Arts (78)	66	76
	Adults (65)	61	65
	Recreation (84)	70	75
	Computer (57)	53	56
	Games (42)	38	42
	Health (60)	54	55
	Home (102)	82	93
	Kids & Teens (37)	31	36
Ranking (393)	Top (288)	235	277
	Bottom (105)	100	104
Total (1,203)		1,031 (86%)	1,135 (94%)

How to Determine Whether a Website is PCI Compliant? It is not easy to directly confirm whether a website is DSS compliant or not, unless the website actively advertises this information. While some cloud and service providers (*e.g.*, Google Cloud [39], Amazon Connect [12], Shopify [38], and Akamai [37]) advertise their PCI compliance status, not all of them disclose such information. However, as e-commerce websites need to show their DSS compliance in order to work with acquirer banks (described in Section 2), it is reasonable to assume that most websites we evaluated have successfully passed the external scanning.

Website Selection. We use two different ways to select websites to increase diversity. First, we downloaded 2,000 Alexa top websites under 10 categories (200 websites per category) to observe security compliance differences based on categories. In Table 4, we show the category-wise breakdown. Among them, we manually identified 810 websites that make payment card transactions. This step is time-consuming and usually requires manually visiting multiple pages (*e.g.*, one needs to visit multiple pages to get to the payment page on *nytimes.com*). Second, to cover websites of different popularity levels, we further select the top 500 and bottom 500 websites (1,000 in total) from Alexa top 1 million website list. We found 288 websites from the top list and 105 websites from the bottom list that accept payment card information (and do not overlap with the previous 811 websites). In total, 1,203 payment-cards-taking websites are selected for scanning by PciCheckerLite.

Findings of E-commerce Website Compliance. 68 websites fully passed our PciCheckerLite test, including the aforementioned cloud providers (Google Cloud, Amazon Connect, Shopify). Our results also confirm that a number of actively operating websites do not comply with the PCI Data Security Standard. As shown in Table 4, out of the 1,203 websites, 1,135 (94%) have at least one vulnerability. More importantly, 1,031 (86%) sites have at least one vulnerability that belong to the "must-fix" vulnerabilities which should have disqualified them as non-compliant. Among them, 520 (43%) sites even have two or more must-fix vulnerabilities.

Then as shown in Table 5, the shopping category has the lowest percentage (87%) of vulnerable websites, while all other categories have a percentage of over 90%. We found several types of high-risk and medium-risk vulnerabilities, including leaving the Mysql port (3306) open, using self-signed or expired certificates, wrong hostnames in the certificate, enabling HTTP TRACE method, and using vulnerable OpenSSH (7.5 or earlier). Supporting TLS v1.0 (low-risk level) is another most common vulnerability we detected (Test case 18), likely due to the need for backward compatibility. SSLv3.0 and TLSv1.0 are known to have multiple man-in-the-middle vulnerabilities [40] and the PCI standard recommends that all web servers and clients must transition to TLSv1.1 or above.

The vulnerabilities in these websites suggest the PCI scanners used by the websites are inadequate and failed to detect the vulnerabilities during the certification scans. Another possibility is that the acquiring banks did not sufficiently examine the merchants' quarterly security reports, allowing merchants to operate without sending adequate security reports to banks as required.

Vulnerable Websites. Below, we highlight some interesting findings without explicitly mentioning the names of vulnerable sites.

Mysql open ports. 59 websites expose the MySql service for remote access. For example, two Slovenian websites that sell healthcare products and car components and a Russian website that sells furnaces and stoves all have this vulnerability. We did not detect any use of default user (root) or no password.

Insecure certificates (self-signed, expired, and insecure modulus). The use of certificates with wrong hostnames (Figure 3 in Appendix) is an issue that appears in 3% of the websites. For some websites, the root cause is not properly configuring HTTPS. For example, one website accepts payment for donations. Since it does not correctly set up HTTPS, it uses a default certificate[5] for HTTPS (Figure 4 in Appendix). In some cases, the websites use HTTPS for payment only while other sensitive content (*i.e.*, items and the cart) are still sent over HTTP. Because the original domain is not properly configured to use HTTPS, it presents the default expired certificate (Figure 5 in Appendix).

Comparison with Existing Tool. Finally, we experimentally compared PciCheckerLite with the state-of-the-art web scanner. Note that existing scanners typically have aggressive pentesting components that are not suitable to test live websites. For this experiment, we choose w3af and have to adapt it to a "non-intrusive low-interactive" version. More specifically, we modify w3af to 1) block intrusive tests (*e.g.*, XSS, SQL injections), 2) disable URL fuzzing, and 3) disable the liveliness testing. For scalability, we also utilized w3af's programmable APIs (w3af_console) to discard the graphic user interface. We call this version as *customized w3af*. For comparison, we ran PciCheckerLite and the customized w3af on 100 websites random from the 1203 sites (in Table 5). For reference, we also ran both tools on our BuggyCart.

The results are shown in Table 6. First, we observe that our system outperforms w3af on Buggycart by detecting all the vulnerabilities. Second, on the 100 real-world websites, our system also detected more *truly vulnerable* websites. Even though w3af flagged more websites (*e.g.*, Test case 7, 29), manually analysis shows that a large portion of the alerts are false positives. For example, under

[5] A self-signed certificate comes with the webserver installation.

Table 5: Testing results on 1,203 real-world websites that accept payment card transactions as of May 3, 2019. We reuse the index numbers of the test cases from Table 3.

Reqs.	Test Cases	Severity	Category (810)										Ranking (393)		Total (1,203)
			Biz. (122)	Shop. (163)	Arts (78)	Adlt. (65)	Recr. (84)	Comp. (57)	Game. (42)	Hlth. (60)	Home. (102)	Kids. (37)	Top (288)	Btm. (105)	
1.2	2. Mysql port (3306) detection	Medium	3	6	4	2	6	2	3	2	4	0	0	27	59 (5%)
	3. OpenSSH available	Low	6	15	11	4	13	6	7	8	12	1	6	27	116 (10%)
2.1	5. Default Mysql user/passwd	High	0	0	0	0	0	0	0	0	0	0	0	0	0 (0%)
2.3	7. Sensitive info over HTTP	High	12	10	12	10	17	10	8	6	10	5	47	22	169 (14%)
4.1	12. Selfsigned cert presented	High	0	0	3	0	1	0	0	1	1	0	0	3	9 (1%)
	13. Weak Cipher Supported	High	0	0	0	0	0	0	0	0	0	0	0	0	0 (0%)
	14. Expired cert presented	Medium	0	0	2	0	2	0	0	1	0	0	0	2	7 (1%)
	15. Wrong hostname in cert	Medium	3	1	3	0	6	2	0	2	4	1	0	10	32 (3%)
	16. Insecure Modulus	High	0	0	0	0	0	0	0	0	0	0	0	1	1 (0.1%)
	17. Weak hash in cert	High	0	0	0	0	0	0	0	0	0	0	0	0	0 (0%)
	18. TLSv1.0 Supported	Low	67	73	53	42	41	40	28	30	67	16	216	71	744 (62%)
6.1	19. OpenSSH vulnerable	High	6	14	11	4	13	6	6	8	11	1	6	26	112 (9%)
6.5	25. Missing script integrity check	Medium	92	109	54	44	44	32	27	42	66	21	154	75	760 (63%)
6.6	29. Server Info available	Medium	26	34	17	17	22	15	17	17	25	11	33	22	256 (21%)
	30. Browsable Dir Enabled	High	0	0	0	0	0	0	0	0	0	0	0	0	0 (0%)
	31. HTTP TRACE supported	High	6	4	3	3	2	5	2	2	6	0	4	6	43 (4%)
	33. Security Headers missing	Medium	18	38	9	12	14	21	9	7	14	7	114	13	276 (23%)

Table 6: Comparison between PciCheckerLite and the `customized w3af` on 100 randomly chosen websites and the Buggycart testbed. We report the number of vulnerable websites detected and the false positives (FP) among them.

Vulnerabilities	100 Random websites		Buggycart	
	Ours (FP)	w3af (FP)	Ours	w3af
2. Mysql port (3306) detection	5 (0)	0 (0)	✓	✗
3. OpenSSH available	10 (0)	0 (0)	✓	✗
5. Default Mysql user/pass	0 (0)	0 (0)	✓	✗
7. Sensitive info over HTTP	12 (0)	27 (17)	✓	✓
12. Selfsigned cert presented	2 (0)	2 (0)	✓	✓
13. Weak Cipher Supported	0 (0)	0 (0)	✓	✗
14. Expired cert presented	0 (0)	3 (3)	✓	✓
15. Wrong hostname in cert	3 (0)	2 (1)	✓	✓
16. Insecure Modulus	0 (0)	0 (0)	✓	✗
17. Weak hash in cert	0 (0)	0 (0)	✓	✗
18. TLSv1.0 Supported	63 (0)	0 (0)	✓	✗
19. OpenSSH vulnerable	10 (0)	0 (0)	✓	✗
25. Missing script integrity check	72 (1)	55 (10)	✓	✓
29. Server Info available	19 (0)	81 (62)	✓	✓
30. Browsable Dir Enabled	0 (0)	0 (0)	✓	✗
31. HTTP TRACE supported	6 (0)	6 (6)	✓	✓
33. Security Headers missing	30 (0)	0 (0)	✓	✗

Test case 7, w3af flags a website if Port 80 is open, while PciCheckerLite reports a website only if the request is not automatically redirected to Port 443 (HTTPS). This design of w3af produces 17 false positives. Under Test case 15, w3af flags a website that uses the certificate for its subdomains (which is not a violation). For Test case 29, w3af flags websites that expose non-critical information whereas we only flag the exposure of exploitable information (*e.g.*, server and framework version numbers). Note that among all vulnerabilities, we only have one FP under Test case 25. This website is flagged by PciCheckerLite for loading Javascript from a third-party domain without an integrity check. Manually analysis shows that the third-party domain and the original website are actually owned by the same organization. Technically, such information is beyond what PciCheckerLite can collect.

6 Disclosure and Discussion

Responsible Disclosure. We have fully disclosed our findings to the PCI Security Standard Council. In May 2019, we shared our paper with the PCI SSC, and successfully got in touch with an experienced member of the Security Council. Through productive exchanges with them, we gained useful insights. *i)* The Security Council shared a copy of our paper to the dedicated companies that host the PCI certification testbeds, who are now aware of our findings; *ii)* Preventing scanners from *gaming the test* is one of their priorities, for example, by constantly updating their testbeds and changing the tests; *iii)* Low interaction constraints make it difficult to test some vulnerabilities externally (which we also experienced and aimed to address in our work); *iv)* The Security Council routinely removes scanners from the ASV list or warns scanners based on the feedback sent by ASV consumers; *v)* Their testbeds exclude vulnerabilities whose CVSS scores are lower than 4; *vi)* Payment brands and acquirer banks need efficient (and automatic) solutions to inspect PCI DSS compliance reports. Insights *ii)*, *iii)*, and *vi)* present interesting research opportunities. In addition, we are in the process of contacting vulnerable websites. Some notifications have been sent out to those that failed test case 2 (open Mysql port) or 19 (vulnerable OpenSSH). Incidentally, we found a few websites have already fixed their problems, for example Netflix upgraded the vulnerable SSH-2.0-OpenSSH_7.2p2 (current Netflix.com server does not show a version number).

Is Improving PCI Certification a Practical Task? From the economics point of view, the concept of for-profit security certification companies may seem like an oxymoron. Intuitively, a scanning vendor might make more money if its scanner is less strict, allowing websites to easily pass the DSS certification test. On the contrary, a

company offering rigorous certification scanning might lose customers when they become frustrated from failing the certification test. Phenomena with misaligned incentives widely exist in many security domains (*e.g.*, ATM security, network security) [41]. Fortunately, unlike the decentralized Internet, PCI security is centrally supervised by the PCI Security Council. Thus, the Council, governing the process of screening and approving scanner vendors, is a strong point of quality control. The enforcement can be strengthened through technical means. Thus, improving the PCI security certification, unlike deploying Internet security protocols [69], is a practical goal that is very reachable in the near future.

Gaming-resistant Self-evolving Testbeds and Open-Source PCI Scanners. A testbed needs to constantly evolve, incorporating new types of vulnerabilities and relocating existing vulnerabilities over time. A fixed testbed is undesirable, as scanners may gradually learn about the test cases and trivially pass the test without conducting a thorough analysis. Automating this process and creating self-evolving testbeds are interesting open research problems.

Competitive open-source PCI/web scanners from non-profit organizations could drive up the quality of commercial vendors, forcing the entire scanner industry to catch up, and providing alternative solutions for merchants to run sanity check on their services. Currently, there are not many high-quality, open-source and deployable web scanners; `w3af` and `ZAP` are among the very few available.

Automate the Workload at Payment Brands and Acquirer Banks. Payment brands and acquirer banks are the ultimate gatekeepers in the PCI DSS enforcement chain. Manually screening millions of scanning reports and questionnaires every quarter is not efficient (and is likely not being done well in practice). Indeed, our real-world experiments suggest that the gatekeeping at the acquirer banks and payment brands appears weak. Thus, automating the report processing for scalable enforcement is urgently needed.

Scanning vs. Self-assessment Questionnaires. There are four major types of Self-assessment Questionnaires or SAQs (A to D) [29]. The different SAQs are designed for different types of merchants, as illustrated in Figure 6 in the Appendix. In SAQs, all the questions are close ended, *i.e.*, multiple choices. For a vast majority of the merchants, the current compliance checking largely relies on the trust of a merchant's honesty and capability of maintaining a secure system. This observation is derived from our analysis of the 340 questions in the self-assessment questionnaire (SAQ) D-Mer, which is an SAQ designed for merchants that process or store cardholder data. Consequently, it is the most comprehensive questionnaire.

We manually went through all the questions the in Self-Assessment Questionnaire (SAQ) D-Mer and categorized them into the five major groups, *network security*, *system security*, *application security*, *application capability*, and *company policies*. 271 of the 340 questions fall under the category of company policies and application capability, where none of them can be automatically verifiable by an external entity (*e.g.*, ASV/web scanners). Only 31 out of the 69 questions on network, system and application security are automatically verifiable by a PCI scanner.

Legal Consequences of Cheating in PCI Certification. The PCI DSS standard is not required by the U.S. federal law. Some state laws do refer to PCI DSS (*e.g.*, Nevada, Minnesota, Washington) [58], stating that merchants must be PCI compliant. However, there is no mentioning about any legal consequences of cheating in the PCI DSS certification process. Thus, it appears that being untruthful when filling out the self-assessment questionnaire would not have any direct legal consequences. The only potential penalty would be an "after effect". For example, a merchant may be fined by the card brand if a data breach happens due to its non-compliance [3].

Limitations. Our work has a few limitations. First, we only tested 6 PCI scanners and 4 web scanners. Given the high expense to order PCI and web scanning, it is unlikely that such an experiment can truly scale up. For PCI scanning, we have tried to increase the diversity of scanner selection by selecting from different price ranges. The website scanners are added to further increase diversity. Second, our paper primarily focuses on the PCI compliance certification of e-commerce websites. Although we did not evaluate the compliance of *banks* (which report to card brands), we argue that it is the same set of the approved PCI scanners that provide the compliance reports for both merchants and banks. The problem revealed in our study should be generally applicable. Third, we did not test vulnerabilities that are not yet covered by the current Data Security Standards (DSS). Future work can further study the comprehensiveness of DSS. Finally, in Section 5, we only tested 1,203 e-commerce websites because it requires manual efforts to verify whether a website accepts payment card information. It is difficult to automate the verification process since one often needs to register an account and visit many pages before finding the payment page. We argue that our experiment already covers websites from various categories and ranking ranges, which is sufficient to demonstrate the prevalence of the problem.

7 Related Work

Website Scanning. The detection of web application vulnerabilities has been well studied by researchers [45, 49, 73]. In [45, 74], authors measured the performance of several black-box web scanners and reported a low detection rate for XSS and SQL injection attacks. The main challenge is to exhaustively discover various web-app states by observing the input/output patterns. Duchene *et al.* [53] proposed an input fuzzer to detect XSS vulnerabilities. Doupé *et al.* [49] proposed to guide fuzzing based on the website's internal states. In [64], authors proposed a black-box method to detect logical flaws using network traffic. In [73], authors used a taint-tracking based detection of XSS vulnerabilities at the client-side. In [65], authors used dynamic execution trace-based behavioral models to detect CSRF vulnerabilities. Although most defenses against XSS and SQL inject attacks prescribe input sanitization [44, 57, 59], in [51], authors proposed an application-agnostic rewrite technique to differentiate scripts from other HTML inputs. We argue that similar research efforts could make a positive impact to the PCI community by (1) producing and releasing high-quality open-sourced tools; and (2) customizing a non-intrusive version of the tool for testing production websites in the PCI DSS context.

Proactive Threat Measurements. Honeypots [62, 66] are useful to collect empirical data on attackers (or defenders). In [56], authors measure attack behaviors by deploying vulnerable web servers waiting to compromised. In [63], authors deployed phishing websites to measure the timeliness of browsers' blacklist mechanisms. In [48], authors measure the capability of the web hosting providers to detect compromised websites by deploying vulnerable websites

within those web hosting services. Our testbed can be regarded as a specialized honeypot to assess the capability of PCI scanners.

Physical Card Frauds. Payment card frauds at ATM or point-of-sale (POS) machines have been studied for decades [42, 43, 47, 52, 61, 70, 71]. Most of these frauds occur due to stealing payment card information during physical transactions [35, 42], and cloning magnetic stripe cards [70, 71]. EMV cards are known to be resistant to card cloning, but are vulnerable to tempered terminals [52], or due to protocol-level vulnerabilities [61] and implementation flaws [47]. Recently, researchers proposed mechanisms to detect magnetic card skimmers [46, 70].

Digital Card Frauds. In the *online* setting, the danger of using *magnetic-stripe-like transactions* is known for years [8, 21]. Various methods (*e.g.*, 3D-Secure [24], Tokenization framework [13]) have been proposed to fix it. Unfortunately, 3D-Secure is found to be inconvenient and easy to break [60]. Tokenization framework offers a great alternative by replacing original card information with temporary tokens during a transaction. However, card information can still be stolen during account setup phase at a poorly secured merchant. Other unregulated digital financial services are also reported to be insecure [68]. In [68], the authors showed that *branchless banking* apps that leverage cellular networks to send/receive cashes are also vulnerable due to flaws such as skipping SSL/TLS certificate validation, and using insecure cryptographic primitives.

8 Conclusion

Our study shows that the PCI data security standard (PCI DSS) is comprehensive, but there is a big gap between the specifications and their real-world enforcement. Our testbed experiments revealed that the vulnerability screening capabilities of some approved scanning vendors (ASV) are inadequate. 5 of the 6 PCI scanners are not compliant with the ASV scanning guidelines. All 6 PCI scanners would certify e-commerce websites that remain vulnerable. Our measurement on 1,203 e-commerce websites shows that 86% of the websites have at least one type of vulnerability that should disqualify them as non-compliant. Our future work is to a design minimum-footprint black-box scanning method.

9 Acknowledgment

This project was supported in part by NSF grants CNS-1717028, CNS-1750101 and OAC-1541105, ONR Grant ONR-N00014-17-1-2498.

References

[1] Common Vulnerability Scoring System Calculator Version 3. https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator. [Online; accessed 28-Aug-2019].
[2] The owasp zed attack proxy (zap). https://www.zaproxy.org/.
[3] Q15: What are the penalties for non-compliance? https://www.pcicomplianceguide.org/faq/#15.
[4] W3af. http://w3af.org/.
[5] SQL injection with raw MD5 hashes (leet more ctf 2010 injection 300). http://cvk.posthaven.com/sql-injection-with-raw-md5-hashes, 2010.
[6] Payment Card Industry (PCI) Point-to-Point Encryption: Solution Requirements and Testing Procedures. https://www.pcisecuritystandards.org/documents/P2PE_v2_r1-1.pdf, 2015.
[7] Payment Card Industry (PCI) Token Service Providers: Additional Security Requirements and Assessment Procedures for Token Service Providers (EMV Payment Tokens). https://www.pcisecuritystandards.org/documents/PCI_TSP_Requirements_v1.pdf, 2015.
[8] All About Fraud: How Crooks Get the CVV. https://krebsonsecurity.com/2016/04/all-about-fraud-how-crooks-get-the-cvv/, 2016. [Online; accessed 8-Jan-2019].
[9] Payment Card Industry (PCI) Payment Application Data Security Standard: Requirements and Security Assessment Procedures. https://www.pcisecuritystandards.org/documents/PA-DSS_v3-2.pdf, 2016.
[10] Payment Card Industry (PCI) PIN Transaction Security (PTS) Hardware Security Module (HSM): Modular Security Requirements. https://www.pcisecuritystandards.org/documents/PCI_HSM_Security_Requirements_v3_2016_final.pdf, 2016.
[11] PCI Self-Assessment Questionnaire Instructions and Guidelines. version 3.2. https://www.pcisecuritystandards.org/documents/SAQ-InstrGuidelines-v3_2.pdf, 2016.
[12] Amazon Connect is Now PCI DSS Compliant. https://aws.amazon.com/about-aws/whats-new/2017/07/amazon-connect-is-now-pci-dss-compliant/, 2017.
[13] EMV Payment Tokenisation Specification: Technical Framework. https://www.emvco.com/terms-of-use/?u=/wp-content/uploads/documents/EMVCo-Payment-Tokenisation-Specification-Technical-Framework-v2.0-1.pdf, 2017.
[14] Giant equifax data breach: 143 million people could be affected. https://money.cnn.com/2017/09/07/technology/business/equifax-data-breach/index.html, 2017.
[15] How many e-commerce companies are there? What's the global e-commerce market size? http://blog.pipecandy.com/e-commerce-companies-market-size/, 2017.
[16] Payment Card Industry 3-D Secure (PCI 3DS): Security Requirements and Assessment Procedures for EMV 3-D Secure Core Components: ACS, DS, and 3DS Server. https://www.pcisecuritystandards.org/documents/PCI-3DS-Core-Security-Standard-v1.pdf, 2017.
[17] Payment Card Industry (PCI) Card Production and Provisioning: Logical Security Requirements. https://www.pcisecuritystandards.org/documents/PCI_Card_Production_Logical_Security_Requirements_v2.pdf, 2017.
[18] Payment Card Industry (PCI) Card Production and Provisioning: Physical Security Requirements. https://www.pcisecuritystandards.org/documents/PCI_Card_Production_Physical_Security_Requirements_v2.pdf, 2017.
[19] Payment Card Industry (PCI) Data Security Standard Approved Scanning Vendor. program guide. version 3.1. https://www.pcisecuritystandards.org/documents/ASV_Program_Guide_v3.1.pdf, 2017.
[20] Approved scanning vendors. https://www.pcisecuritystandards.org/assessors_and_solutions/approved_scanning_vendors, 2018.
[21] Card Fraud on the Rise, Despite National EMV Adoption. https://geminiadvisory.io/card-fraud-on-the-rise/, 2018. [Online; accessed 8-Jan-2019].
[22] Cardconnect: A new wave of payment processing. https://cardconnect.com/, 2018.
[23] A Comprehensive Guide to PCI DSS Merchant Levels. https://semafone.com/blog/a-comprehensive-guide-to-pci-dss-merchant-levels/, 2018.
[24] EMV 3-D Secure: Protocol and Core Functions Specification. https://www.emvco.com/wp-content/uploads/documents/EMVCo_3DS_Spec_v220_122018.pdf, 2018.
[25] Let's Encrypt. https://letsencrypt.org/, 2018.
[26] Opencart. https://www.opencart.com/, 2018.
[27] Payment Card Industry (PCI) Data Security Standard: Requirements and security assessment procedures. https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf, 2018.
[28] Payment Card Industry (PCI) Data Security Standard Self-Assessment Questionnaire D and Attestation of Compliance for Merchants: All other SAQ-Eligible Merchants. https://www.pcisecuritystandards.org/documents/PCI-DSS-v3_2_1-SAQ-D_Merchant.pdf?agreement=true&time=1557603304233, 2018.
[29] Payment Card Industry (PCI) Data Security Standard Self-Assessment Questionnaire: Instructions and Guidelines. https://finance.ubc.ca/sites/finserv.ubc.ca/files/banking-leases/PCI_DSS_SAQ_Instructions_Guidelines.pdf, 2018.
[30] Payment Card Industry (PCI) PIN Transaction Security (PTS) Point of Interaction (POI): Modular Security Requirements. https://www.pcisecuritystandards.org/documents/PCI_PTS_POI_SRs_v5-1.pdf, 2018.
[31] Payment Card Industry (PCI) Software-based PIN Entry on COTS: Security Requirements. https://www.pcisecuritystandards.org/documents/SPoC_Security__Requirements_v1.0.pdf, 2018.
[32] Who's In Your Online Shopping Cart? https://krebsonsecurity.com/2018/11/whos-in-your-online-shopping-cart/, 2018.
[33] BlueCrypt: Cryptographic Key Length Recommendation. https://www.keylength.com/en/4/, 2019.
[34] DB-Engines Ranking. https://db-engines.com/en/ranking, 2019.
[35] Insert Skimmer + Camera Cover PIN Stealer. https://krebsonsecurity.com/2019/03/insert-skimmer-camera-cover-pin-stealer/, 2019. [Online; accessed 20-Mar-2019].
[36] netcat. https://en.wikipedia.org/wiki/Netcat, 2019.
[37] PCI DSS Compliance. https://www.akamai.com/us/en/resources/pci-dss-compliance.jsp, 2019.
[38] Shopify meets all 6 categories of PCI standards. https://www.shopify.ca/security/pci-compliant, 2019.
[39] Standards, Regulations & Certifications. https://cloud.google.com/security/compliance/pci-dss/, 2019.

[40] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Béguelin, S. Z., and Zimmermann, P. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2015).
[41] Anderson, R., and Moore, T. The economics of information security. *Science 314*, 5799 (2006), 610–613.
[42] Anderson, R. J. Why cryptosystems fail. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)* (1993).
[43] Anderson, R. J., and Murdoch, S. J. EMV: why payment systems fail. *Commun. ACM 57*, 6 (2014), 24–28.
[44] Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C., and Vigna, G. Saner: Composing static and dynamic analysis to validate sanitization in web applications. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (2008).
[45] Bau, J., Bursztein, E., Gupta, D., and Mitchell, J. C. State of the art: Automated black-box web application vulnerability testing. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (2010).
[46] Bhaskar, N., Bland, M., Levchenko, K., and Schulman, A. Please pay inside: Evaluating bluetooth-based detection of gas pump skimmers. In *Proceedings of the 28th USENIX Security Symposium (USENIX SEC)* (2019).
[47] Bond, M., Choudary, O., Murdoch, S. J., Skorobogatov, S. P., and Anderson, R. J. Chip and skim: Cloning EMV cards with the pre-play attack. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (2014).
[48] Canali, D., Balzarotti, D., and Francillon, A. The role of web hosting providers in detecting compromised websites. In *Proceedings of the International World Wide Web Conference (WWW)* (2013).
[49] Doupé, A., Cavedon, L., Kruegel, C., and Vigna, G. Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner. In *Proceedings of the USENIX Security Symposium (USENIX SEC)* (2012).
[50] Doupé, A., Cova, M., and Vigna, G. Why Johnny Can't Pentest: An Analysis of Black-Box Web Vulnerability Scanners. In *Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)* (2010).
[51] Doupé, A., Cui, W., Jakubowski, M. H., Peinado, M., Kruegel, C., and Vigna, G. dedacota: toward preventing server-side XSS via automatic code and data separation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2013).
[52] Drimer, S., and Murdoch, S. J. Keep your enemies close: Distance bounding against smartcard relay attacks. In *Proceedings of the USENIX Security Symposium (USENIX SEC)* (2007).
[53] Duchene, F., Rawat, S., Richier, J., and Groz, R. Kameleonfuzz: evolutionary fuzzing for black-box XSS detection. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (CODASPY)* (2014).
[54] Gamero-Garrido, A., Savage, S., Levchenko, K., and Snoeren, A. C. Quantifying the pressure of legal risks on third-party vulnerability research. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2017).
[55] Grossman, J. Cross site tracing (xst). https://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf.
[56] Han, X., Kheir, N., and Balzarotti, D. Phisheye: Live monitoring of sandboxed phishing kits. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2016).
[57] Hooimeijer, P., Livshits, B., Molnar, D., Saxena, P., and Veanes, M. Fast and precise sanitizer analysis with BEK. In *Proceedings of the USENIX Security Symposium (USENIX SEC)* (2011).
[58] Kemp, T. Buckle up with Cybersecurity ... It's the law. https://www.forbes.com/sites/tomkemp/2012/02/01/buckle-up-with-cybersecurity-its-the-law/#5d83d3a31d72, 2012.
[59] Livshits, B., and Chong, S. Towards fully automatic placement of security sanitizers and declassifiers. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)* (2013).
[60] Murdoch, S. J., and Anderson, R. J. Verified by visa and mastercard securecode: Or, how not to design authentication. In *Proceedings of the International Conference on Financial Cryptography and Data Security (FC)* (2010).
[61] Murdoch, S. J., Drimer, S., Anderson, R. J., and Bond, M. Chip and PIN is broken. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (2010).
[62] Nazario, J. PhoneyC: A virtual client honeypot. In *Proceedings of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)* (2009).
[63] Oest, A., Safaei, Y., Doupé, A., Ahn, G.-J., Wardman, B., and Tyers, K. Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (2019).
[64] Pellegrino, G., and Balzarotti, D. Toward black-box detection of logic flaws in web applications. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)* (2014).
[65] Pellegrino, G., Johns, M., Koch, S., Backes, M., and Rossow, C. Deemon: Detecting CSRF with dynamic analysis and property graphs. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)* (2017).
[66] Provos, N. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium (USENIX SEC)* (2004).
[67] Ramos, S. H., Villalba, M. T., and Lacuesta, R. MQTT Security: A Novel Fuzzing Approach. *Wireless Communications and Mobile Computing 2018* (2018).
[68] Reaves, B., Scaife, N., Bates, A., Traynor, P., and Butler, K. R. B. Mo(bile) money, mo(bile) problems: Analysis of branchless banking applications in the developing world. In *Proceedings of the USENIX Security Symposium (USENIX SEC)* (2015).
[69] Routing security for policymakers: An Internet society white paper, October 2018. Internet Society. https://www.manrs.org/wp-content/uploads/2018/10/Routing-Security-for-Policymakers-EN.pdf.
[70] Scaife, N., Peeters, C., and Traynor, P. Fear the Reaper: Characterization and Fast Detection of Card Skimmers. In *Proceedings of the USENIX Security Symposium (USENIX SEC)* (2018).
[71] Scaife, N., Peeters, C., Velez, C., Zhao, H., Traynor, P., and Arnold, D. P. The cards aren't alright: Detecting counterfeit gift cards using encoding jitter. In *Proceedings of the IEEE Symposium on Security and Privacy, (S&P)* (2018).
[72] Shu, X., Tian, K., Ciambrone, A., and Yao, D. Breaking the target: An analysis of target data breach and lessons learned. *CoRR abs/1701.04940* (2017).
[73] Steffens, M., Rossow, C., Johns, M., and Stock, B. Don't trust the locals: Investigating the prevalence of persistent client-side cross-site scripting in the wild. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019* (2019).
[74] Vieira, M., Antunes, N., and Madeira, H. Using web security scanners to detect vulnerabilities in web services. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2009).

Appendix

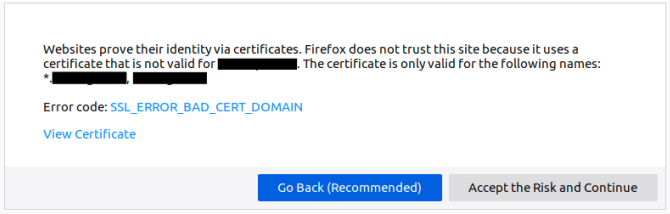

Figure 3: An example of wrong hostname in the certificate. The domain (*a****.***) uses a certificate that is issued for a different domain name (**.n*******.***).**

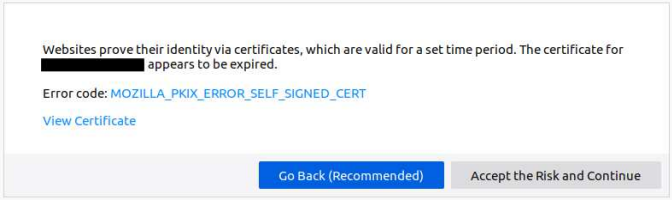

Figure 4: Self-signed certificate used by (*r****.***), a website that accepts payment cards for donations.**

Implementation Details of PciCheckerLite

PciCheckerLite follows a series of rules for vulnerability testing. The index of the rules matches with the testing cases discussed in the paper. As described in the paper, we only focus on a subset of test cases that do not disrupt or cause any negative impact to the remote servers (for ethical considerations). The implementation details are as follows.

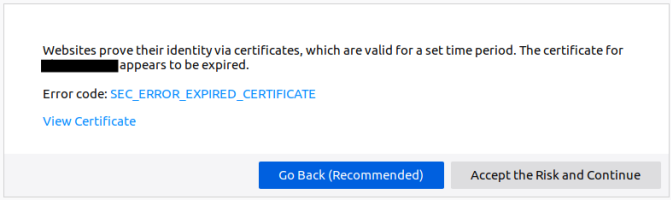

Figure 5: ($u^{*******.***}$) uses expired certificates by default and redirects users to a secure sub-domain with proper certificate during payment.

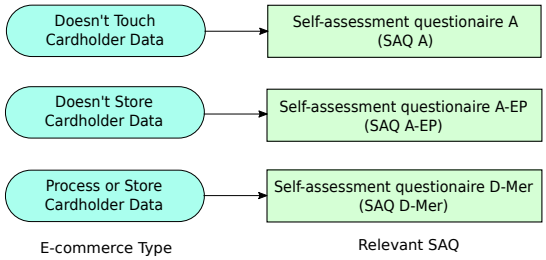

Figure 6: Self-Assessment Questionnaires (SAQs) for different types of e-commerce merchants.

Rule 2. Database port detection. For database port detection, we choose to probe for Mysql port[6]. The reason for choosing Mysql port are *i)* Mysql is among the top three (Mysql, Oracle, Microsoft SQL Server) most popular databases in the world [34]; *ii)* Mysql is free; and *iii)* it supports a wide range of programming languages. The access to Mysql port (*e.g.*, 3306) is disabled by default. It is very dangerous to enable remote access to Mysql database for an arbitrary client. We check the Mysql port using *nc* [36], which is a Unix utility tool that reads and writes data across network connections using the TCP or UDP protocol.

Rule 5. Default Mysql user/password detection. If the Mysql database of a website is remotely accessible, we further check for the default username and password. A typical Mysql installation has a user "*root*" with an empty password, unless it is otherwise customized or disabled. As such, we run a Mysql client to connect to the remote host using the default username and password. PciCheckerLite terminates the connection immediately and raises an alert if the attempt is successful.

Rules 3 & 19. Checking OpenSSH's availability and version. We use *nc* [36] to connect with port 22 of the remote OpenSSH server. If OpenSSH runs on port 22, then it will return the server information (*e.g.*, OpenSSH version, OS type, OS version). We parse the returned information to determine the version of the OpenSSH server. We consider any installation versions before *OpenSSH_7.6* as vulnerable.

Rules 29 & 33. Checking HTTP header information. Extracting HTTP information does not require the rich browser functionality. We use Java net URL APIs to open HTTP connections for extracting HTTP headers. For case 29, we raise a warning only if we detect that the "`Server`" header contains server name and version. For case 33, we raise a warning if any of the four security header (*i.e.*, `X-Frame-Options`, `X-XSS-Protection`, `Strict-Transport-Security`, `X-Content-Type-Options`) is missing.

Rule 7. Sensitive information over HTTP. We tested whether all the HTTP traffic is redirected to HTTPS by default. We open an HTTP connection with the server and follow the redirection chain. If the server doesn't redirect to HTTPS, we raise an alert. We use Java net URL APIs to implement this test case.

Rules 18 & 13. TLSv1.0 and weak cipher negotiation. We use OpenSSL's *s_client* tool to establish a SSL/TLS connection using TLSv1.0 protocol. PciCheckerLite raises a warning if the connection is successful. We also use *s_client* to negotiate the ciphersuite with the remote server. PciCheckerLite raises a warning if we successfully negotiate with a ciphersuite that contains a weak cipher (*i.e.*, IDEA, DES, MD5).

Rules 12, 14, 15, 16 & 17. Retrieving and examining the certificate. We use OpenSSL's *s_client* tool to retrieve the SSL certificate of a remote server. To parse the certificate, we use APIs from *java.security.cert* package. To check whether a certificate is self-signed (Case 12), we used the public key of the certificate to verify the certificate itself. To check whether the certificate is expired, we use the *checkValidity()* method of *X509Certificate* API (Case 14). If the subject domainname (DN) or any alternate DN of a certificate doesn't match with the server domainname, then PciCheckerLite raises an alert (Case 15). Regarding the public key sizes for factoring modulus (*e.g.*, RSA, DSA), the discrete logarithm (*e.g.*, Diffie-Hellman), and the elliptic curve (*e.g.*, ECDSA) based algorithms, NIST recommends them to be 2048, 224 and 224 bits, respectively [33]. PciCheckerLite raises alert if the key size is smaller than what is recommended (Case 16). If the signing algorithm uses any of the weak hashing algorithms (*e.g.*, MD5, SHA, SHA1, SHA-1), PciCheckerLite raises warnings (Case 17).

Rule 25. Script source integrity check. A website is expected to check the integrity of any JavaScript code that is loaded externally to the browser. To enable script source integrity check, a server can use the "*integrity*" attribute of the *script* tag. In the "*integrity*" attribute, the server should mention the hashing algorithm and the hash value of the script that should be used to check the integrity. PciCheckerLite downloads the index page of a website. After that, it collects all the *script* tags, and checks if the *script* tags contain any external URL (excluding the website's CDN URLs). Then it looks for the *integrity* attribute for the scripts loaded from external URLs, and raises alert if the *integrity* attribute is missing. We only perform this test for the index page (instead of all the pages) of a website to keep the test lightweight. The number of vulnerable websites detected by this test can only be interpreted as a lower bound.

Rule 30. Checking for browsable directories. We check whether the directories are browsable in a website. To avoid redundant traffic, we reuse the collected JavaScript script URLs for case 25. We then examine the common parent directory of all the internal URLs. Finally, we send a GET request to fetch the content of the directory. If directory browsing is enabled, the server will return a response with code 200 with a page containing the listing of files and directories of the specified path. Otherwise, it should return an error

[6]We do not probe for multiple ports to avoid suspicions for possible port scanning. However, a similar technique can be used to probe for other databases.

	Yes	Yes with CCW	No	N/A	Not Tested
The card verification code or value (three digit or four-digit number printed on the front or back of a payment card) is not stored after authorization?	□	□	□	□	□

Figure 7: A sample question from the Self-Assessment Questionnaire D (SAQ D) [28]. "Yes with CCW" means "the expected testing has been performed, the requirement has been met with the assistance of a compensating control, and a Compensating Control Worksheet (CCW) is required to be submitted along with the questionnaire" .

Table 7: A summary of the guidelines for ASV scanners [19]. In the fourth column, we show the categories that are required to be fixed. "*" means that in the SSL/TLS category, all the vulnerabilities are required to be fixed, except case 18.

Target Component	Expectation	Test-cases	Must fix?
Firewalls and Routers	1. Must scan all network devices such as firewalls and external routers. 2. Must test for known vulnerabilities and patches.	1	Yes
Operating Systems	1. Must scan to determine the OS type and version. 2. An unsupported OS must be marked as an automatic failure.	-	Yes
Database Servers	1. Must test for open access to databases from the Internet. 2. If found - must be marked as an automatic failure (Req. 1.3.6)	2	Yes
Web Servers	1. Must be able to test for all known vulnerabilities and configuration issues. 2. Report if directory browsing is observed.	30	Yes
Application Servers	1. Must be able to test for all known vulnerabilities and configuration issues.	29, 33	Yes
Common Web Scripts	1. Must be able to find common web scripts (e.g., CGI, e-commerce, etc.).	-	Yes
Built-in Accounts	1. Look for default username/passwords in routers, firewalls, OS and web or DB servers. 2. Such vulnerability must be marked as an automatic failure. (Req 2.1)	5, 6	Yes
DNS and Mail Servers	1. Must be able to detect the presence 2. Must test for known vulnerabilities and configuration issues 3. Report if a vulnerability is observed (automatic failure for DNS server vulnerabilities).	-	Yes
Virtualization components	1. Must be able to test for all known vulnerabilities	-	Yes
Web Applications	Must find common vulnerabilities (automatically/manually) including the following: 1. Unvalidated parameters that might lead to SQL injection. 2. Cross-site scripting (XSS) flaws 3. Directory traversal vulnerabilities 4. HTTP response splitting/header injection 5. Information leakage: phpinfo(), Insecure HTTP methods, detailed error msg 6. If found any of the above must be marked as an automatic failure	21, 22, 23, 24, 25, 26, 27, 28, 31, 32	Yes
Other Applications	1. Must test for known vulnerabilities and configuration issues	20	Yes
Common Services	1. Must test for known vulnerabilities and configuration issues	19	Yes
Wireless Access Points	1. Must be able to detect wireless access points 2. Must test and report known vulnerabilities and configuration issues	-	Yes
Backdoors/Malware	1. Must test for remotely detectable backdoors/malware 2. Report automatic failure if found one	-	Yes
SSL/TLS	Must find: 1. Various version of crypto protocols 2. Detect the encryption algorithms and encryption key strengths 3. Detect signing algorithms used for all server certificates 4. Detect and report on certificate validity 5. Detect and report on whether CN matches the hostname 6. Mark as failure if supports SSL or early versions of TLS.	12-18	Yes*
Anonymous Key agreement Protocol	1. Must identify protocols allowing anonymous/non-authenticated cipher suites 2. Report if found one	-	Yes
Remote Access	1. Must be able to detect remote access software 2. Must report if one is detected. 3. Must test and report known vulnerabilities and configuration issues	3, 4 19, 20	Yes
Point-of-sale (POS) Software	1. Should look for POS software 2. If found - ask for justification	-	No
Embedded links or code from out-of-scope domains	1. Should look for out-of-scope links/code 2. If found - ask for justification	-	No
Insecure Services/ industry-deprecated protocols	1. If found one - ask for justification	-	No
Unknown services	1. Should look for unknown services and report if found	-	No

response code (*e.g.*, 404 - not found, 403 - Forbidden). This test only determines if a directory is browsable. We never store any of the returned pages during the test.

Rule 31. HTTP TRACE supported. HTTP TRACE method is used for diagnostic purposes. If it is enabled, the web server will respond to a request by echoing in its response the exact request that it has received. In [55], the author has shown that HTTP TRACE can be used to steal sensitive information (*e.g.*, cookie, credentials). To examine the HTTP TRACE configuration, we send a HTTP request by setting the method to `TRACE`. If the TRACE method is enabled by the server, the server will echo the request in the response with a code 200.

Table 8: Specifications defined by the PCI Security Standard Council (SSC) along with their targets, evaluators, assessors and whether it is enforced by SSC. "COTS" stands for Commercial Off-The-Shelf.

PCI Specifications	Target(s)	Evaluator(s)	Assessor(s): Type	Required?
Data Security Standard (DSS) [27]	Merchant, Acquirer Bank, Issuer Bank, Token Service Provider, Service Provider	Acquirer, Payment Brand	QSA: Manual ASV: Automated	Yes
Card Production and Provisioning (CPP) [17, 18]	Card Issuer, Card Manufacturer, Token Service Provider	Payment Brand	CPP-QSA: Manual	Yes
Payment Application DSS (PA DSS) [9]	PA Vendors	PA-QSA	PA-QSA: Manual	Optional
Point-to-Point Encryption (P2PE) [6]	POS Device Vendors	P2PE-QSA	P2PE-QSA: Manual	Optional
PIN Transaction Security (PTS) [10, 30]	PIN Pad Vendors	PTS Labs	PTS Labs: Manual	Optional
3-D Secure (3DS) [16]	3DS Server, 3DS Directory Server, 3DS Access Control Server	Payment Brand	3DS-QSA: Manual	Optional
Software-Based PIN Entry on COTS (SPoC) [31]	PIN-based Cardholder verification method (CVM) Apps	SPoC Labs	SPoC Labs: Manual	Optional
Token Service Provider (TSP) [7]	Token Service Providers	P2PE-QSA	P2PE-QSA: Manual	Optional

Table 9: PCI DSS requirements are presented with expected testing (from SAQ D-Mer) and the potential test-cases that can be used to evaluate the ASV scanning.

No.	Requirement	Expected Testing	Testcase
1.1	Formalize testing when firewall configurations change	1. Review current network diagram 2. Examine network configuration	N/A
1.2	Build a firewall to restrict "untrusted" traffic to cardholder data environment	1. Review firewall and router config 2. Examine firewall and router config	1. Enable/disable firewall.
1.3	Prohibit direct public access between Internet and cardholder data environment	1. Examine firewall and router config	2. Expose Mysql to the Internet 3. SSH over public Internet 4. Remote access to PhpMyadmin
1.4	Install a firewall on computers that have connectivity to the Internet and organization's network	1. Examine employee owned-devices	N/A
2.1	Always change vendor-supplied defaults before installing a System on the network	1. Examine vendor documentations 2. Observe system configurations	5. Use default DB user/password 6. Use default Phpmyadmin user/password
2.2	Develop a configuration standards for all system components that address all known security vulnerabilities.	1. Examine vendor documentations 2. Observe system configurations	N/A
2.3	Encrypt using Strong cryptography all non-console administrative access such as browser/web-based management tools	1. Examine system components 2. Examine system configurations 3. Observe an administrator log on	7. Sensitive information over HTTP
2.4	Shared hosting providers must also comply with PCI DSS requirements	1. Examine system inventory	N/A
3.1	Establish cardholder data retention and disposal policies	1. Review data retention and disposal policies	N/A
3.2	Do not store sensitive authentication data (even it is encrypted)	1. Examine system configurations 2. Examine deletion processes	8. Store CVV in DB
3.3	Mask PAN when displayed	1. Examine system configurations 2. Observe displays of PAN	9. Show unmask PAN
3.4	Render PAN unreadable anywhere it is stored	1. Examine data repositories 2. Examine removable media 3. Examine audit logs	10. Store plain-text PAN (OpenCart)
3.5	Secure keys that are used to encrypt stored cardholder data or other keys	1. Examine system configurations 2. Examine key storage locations	11. Use hardcoded key for encrypting PAN
3.6	Document all key-management process	1. Review key-management procedures	N/A
4.1	Use strong cryptography and security protocols during transmission of cardholder data.	1. Review system configurations	12. Use self-signed certificate 13. Use insecure block cipher 14. Use Expired certificate 15. Use cert. with wrong hostname 16. Use 1024 bit DH modulus. 17. Use weak hash in SSL certificate 18. Use TLSv1.0
4.2	Never send PAN over unprotected user messaging technologies.	1. Review policies and procedures	N/A
5.1	Deploy anti-virus software on all systems	1. Examine system configurations 2. Interview personnel	N/A
5.2	Ensure all anti-virus mechanisms are current, running and generating audit log	1. Examine anti-virus configurations 2. Review log retention process 3. Examine system configurations	N/A
6.1	Ensure that all system components are protected from known vulnerabilities	1. Examine system components 2. Compare the list of security patches	19. Use vulnerable of OpenSSH 20. Use vulnerable PhpMyadmin
6.2	Establish a process to identify and assign risk to newly discovered security vulnerabilities	1. Review policies and procedures	N/A
6.3	Develop software applications in accordance with PCI DSS and industry best practices	1. Review software development process	N/A
6.4	Follow change control processes and procedures for all changes to system components	1. Review change control process	N/A
6.5	Develop applications based on secure coding guidelines and review custom application code	1. Review software-development policies	21. Implant SQL injection in admin login 22. Implant SQL injection in customer login 23. Disable password retry limit 24. Disable restriction on password length. 25. Use JS from external source insecurely 26. Do not hide program crashes 27. Implant XSS 28. Implant CSRF
6.6	Ensure all public-facing applications are protected against known attacks	1. Examine system configuration	29. Present server info in security Headers. 30. Browsable web directories. 31. Enable HTTP Trace/Track 32. Enable phpinfo() 33. Disable security headers
7	Restrict access to cardholder data based on roles	1. Examine access control policy 2. Review vendor documentation 3. Examine system configuration 4. Interview personnel	N/A
8.4[7]	Render all passwords unreadable during storage and transmission for all system components	1. Examine system configuration	34. Store unsalted customer passwords 35. Store plaintext passwords
9	Restrict physical access to cardholder data	1. Observe process 2. Review policies and procedures 3. Interview personnel	N/A
10	Track and monitor all access to network resource and cardholder data	1. Interview personnel 2. Observe audit logs 3. Examine audit log settings	N/A
11	Regularly test security systems and processes	1. Interview personnel 2. Examine scope of testing 3. Review results of ASV scans	N/A
12	Maintain a policy that addresses information security for all personnel	1. Review formal risk assessment 2. Review security policy 3. Interview personnel.	N/A

[7] Other requirements under 8 are not testable.