TLSAssistant Goes FINSEC A Security Platform Integration Extending Threat Intelligence Language

Salvatore Manfredi¹,², Silvio Ranise¹,³, Giada Sciarretta¹, and Alessandro Tomasi¹

¹ Security and Trust, FBK, Trento, Italy
{smmanfredi,ranise,sciarretta,altomasi}@fbk.eu
² DIBRIS, University of Genoa, Genoa, Italy
³ Department of Mathematics, University of Trento, Trento, Italy

Abstract. We present the integration of TLSAssistant, a tool for TLS vulnerability scanning and mitigation, with an online platform of services for cybersecurity in critical infrastructure. We highlight the added value of intelligence sharing and synergies with other services on the platform, as well as the non-trivial challenges encountered in the process.

Keywords: Integrated cybersecurity · Threat intelligence exchange · STIX · TLS

1 Introduction

Sharing threat intelligence, evaluating risk exposure and responding to threats in a timely manner are an open challenge - particularly in complex systems and critical infrastructures. Security platforms are emerging as innovative answers. There is no wide consensus on a common definition of the term security platform. Three crucial capabilities that seem to be recurring in several descriptions [12] are the following: (i) run a number of different security tools simultaneously across as many environments as possible; (ii) integrate the various tools in a uniform solution so to share threat intelligence information, contribute to the timely evaluation of the risk exposure, and participate to respond to threats; (iii) automate as much as possible security operations so to keep up with the sheer number and speed of transactions and attacks in complex systems and critical infrastructures.

While there exist many tools able to provide security-related functionalities, such as penetration testing and intrusion detection, being able to integrate them in a security platform and guarantee the features described above is a non-trivial task. This is so because individual tools are designed in isolation with little or no attention to information sharing in both input and output. This is not surprising as the tools are often developed by different vendors to maximize their effectiveness when used stand-alone.

© Springer Nature Switzerland AG 2021
H. Abie et al. (Eds.): CPS4CIP 2020, LNCS 12618, pp. 16–30, 2021.
https://doi.org/10.1007/978-3-030-69781-5_2
In summary, the main challenges to re-using the wealth of security techniques available in stand-alone tools and effectively integrate them in a security platform are the following:

1. merging heterogeneous sources of security intelligence in a coherent way makes integration a complex task;
2. the results returned by the various tools must be integrated in a coherent and meaningful way to be able to calculate a risk exposure of the system under analysis.

With the arrival of the revised Payment Services Directive (PSD2) [3] and its standardization effort by the Berlin Group [1], the focus on securing TLS has become more and more important and relevant in financial contexts. In this paper, we present the integration of TLSAssistant [25], an open source tool that combines state-of-the-art TLS analyzers with a report system that suggests appropriate mitigations, in the FINSEC platform [5], which is an integrated framework for predictive and collaborative security of financial infrastructures.

The goals and added value in transitioning TLSAssistant from a command-line tool to a cloud service integrated in the FINSEC platform were (i) sharing the long-term mitigation intelligence provided by TLSAssistant reports; (ii) sharing the immediate per-scan intelligence of vulnerabilities sighted at specific URLs; and (iii) potentially enhancing the functionality of other services, based on individual integration. Achieving these goals let us face the two aforementioned challenges. In particular, the first challenge allows us to introduce in TLSAssistant the use of STIX, a language for exchanging cyber threat intelligence to simplify the integration; while the second challenge allows us to introduce the importance of defining appropriate risk metrics and associate them to the vulnerabilities detected by the tools so that they can be integrated and used by the Risk Assessment Engine available in the platform.

Plan of the Paper. Section 2 provides the necessary background notions on TLS, its vulnerabilities and the impact of its use when securing Financial APIs. The section also introduces TLSAssistant, a tool developed to assist system administrator to secure their TLS deployments. Section 3 gives an overview of the FINSEC platform by describing its structure, how it works and the parties involved. Section 4 describes how we addressed the challenges arising from the integration of TLSAssistant in FINSEC, and Sect. 5 further details the integration with the Risk Assessment Engine as deployed in FINSEC. Section 6 concludes the paper and highlights future work.

2 Assisted Hardening of TLS Configuration

Transport Layer Security (TLS) was first introduced in 1999 [15] to provide both confidentiality and integrity between communicating entities. Unfortunately, the secure configuration of TLS instances is far from trivial and requires time and security insight. In this section, we first give a concise description of two known
vulnerabilities (Sect. 2.1); then we present **TLSAssistant**, a tool proposed to assist system administrators and app developers to deploy resilient instances of the TLS protocol (Sect. 2.2); finally we describe the role of TLS for financial services (Sect. 2.3).

### 2.1 Vulnerabilities on TLS

TLS has been updated several times [9,16,17] to improve its security. The latest version, TLS 1.3, removes several now-deprecated cipher suites that made the protocol prone to a large number of vulnerabilities. Nevertheless, the most widely supported version is currently TLS 1.2. On this protocol version or prior, TLS suffers from a wide set of vulnerabilities [26]. In this paper, we focus on two representative examples:

**Bar Mitzvah.** [10] By exploiting the invariance weakness of the RC4 stream cipher, an attacker is able to retrieve the session cookie by guessing the least significant bits of the keystream. RC4 is a very widely used cipher suite. It has been known since 2013 that some RC4 weaknesses also affect SSL/TLS; nevertheless, almost 10% of the most popular sites in the world (based on Alexa’s list) is still supporting it [22].

**ROBOT.** [2] (*Return Of Bleichenbacher’s Oracle Threat*): due to the availability of the PKCS#1v1.5 padding algorithm in RSA, an attacker is able to extract the private key of the session and breaking the message confidentiality. By using an adaptive chosen-ciphertext attack, based on Daniel Bleichenbacher’s chosen-ciphertext attack, the victim is forced to leak information that help the attacker to guess the key. The key can then be used to decrypt HTTPS traffic sent between the TLS server and the user’s browser. Even though the most popular sites in the world are not vulnerable to ROBOT [22], researchers are still discovering different ways to exploit the same vulnerability with small variations to the Bleichenbacher attack, some of them affecting even TLS 1.3 [23].

### 2.2 TLSAssistant

TLSAssistant is an open-source tool [13] proposed to assist system administrators and app developers to deploy resilient instances of the TLS protocol. By bringing together different powerful analyzers, **TLSAssistant** is able to cover a full-range of analysis on all the parties involved in a secure communication and to provide a set of mitigation measures that aim to mitigate the impact of the identified vulnerabilities.

**Architecture.** TLSAssistant is written in Bash and can thus be invoked via command-line. The tool takes as input the target to be evaluated (e.g., the IP address of a server) and outputs a report file. The content of the report depends on the detected weaknesses and on the level of verbosity the user chose.
Figure 1 shows the architecture with its two main components: Analyzer and Evaluator.

Analyzer takes as input the address of the target webserver. TLSAssistant v1.2, the version we integrated in the FINSEC platform, included the following command-line tools: testssl.sh [29], MalloDroid [4] and tlsfuzzer [11], HTTP/HSTS checker. Once loaded, the Analyzer module will run each of the tools related to the required scan, collect their reports and forward them to the Evaluator.

Evaluator is responsible for the enumeration of the detected vulnerabilities and the generation of the report that will guide the system administrator to apply the appropriate mitigations. It is currently able to generate reports that contain the vulnerabilities detected by the Analyzer and the related mitigations described at various levels of abstraction:

- **Textual Description:** natural language description of the TLS vulnerability, identification of the respective CVE ID, CVSS score and related mitigations (brief explanation of the actions to perform).
- **Code Snippet:** a fragment of code that can be copy-pasted into the webserver’s configuration to seamlessly fix the weakness. TLSAssistant can detect any webserver but is currently only able to provide snippets for Apache and nginx HTTP server. Together with the snippet, the report will provide a set of steps on how to find the correct file/line to edit.

For example, in the case of an Apache server vulnerable to Bar Mitzvah, the report will contain the information shown in Fig. 2.

As described in Sect. 4.1, from version v1.2, TLSAssistant is able to export the analysis result in STIX (Structured Threat Information eXpression) language, which supports cyber threat intelligence activities and facilitates integration within security platforms (e.g., FINSEC).

### 2.3 TLS and Financial API

Under the revised Payment Services Directive (PSD2) [3], Account Servicing Payment Service Providers (ASPSP) are to provide an interface for third parties to access account information and perform operations (e.g., payments) on behalf of the account holder.
The Berlin Group standards and harmonization initiative proposes several possible approaches in its detailed “Access to Account (XS2A) Framework” [1], providing a detailed description of RESTful Application Programming Interfaces (APIs) and their usage for the purposes of authentication of involved parties and authorization to access service resources, such as account information, payment initiation, and confirmation of funds. The security of these APIs is based on both the transport and application layers. The first core technology explicitly identified by the guidelines is the TLS protocol: in particular, “the communication between the Third Party Provider (TPP) and the ASPSP is always secured by using a TLS-connection using TLS version 1.2 or higher.” [1].

Recommendations for security and identification standards in XS2A have been published by Open Banking Europe [21], which explicitly assume the use of TLS to guarantee confidentiality and integrity. The deployment of TLS is also assumed by the Financial-grade API (FAPI) specification [19] for authentication and authorization.

While PSD2 APIs are one of the most recent examples of online financial services relying on the security of TLS, there are several other examples such as home banking web and mobile applications; is also explicitly cited as an example of how to comply with the Payment Card Industry Data Security Standard (PCI-DSS) [20] Requirement 4.1, to “Use strong cryptography and security protocols to safeguard sensitive cardholder data during transmission over open, public networks”.

---

**Bar Mitzvah**

By exploiting the invariance weakness of the RC4 stream cipher, an attacker is able to retrieve the session cookie by guessing the LSBS (least significant bits) of the keystream. After a phase in which the attacker sniffs the connection between two parties, it detects a weak key usage and tries to exploit the weakness.

**CVE**: 2015-2808

**CVSSv2 score**: 4.3 (Medium)

**Mitigation**

Disable the RC4 stream cipher.

**Code Snippet**

1. open your Apache configuration file (default: `/etc/apache2/sites-available/default-ssl.conf`);
2. find the line starting with: `SSLCipherSuite`;
3. add the string `:1RC4` at the end.

N.B. restart the server by typing: `sudo service apache2 restart`.

---

*Fig. 2. TLSAssistant report for Bar Mitzvah - Apache webserver.*
Beyond online services, TLS is deployed in many IoT devices [24] and client-end TLS proxies (e.g., in anti-virus products) [28]. Vulnerabilities in their TLS configuration and implementation can significantly downgrade the overall cyber and physical security, affecting many different enterprise sectors. For example, in the financial sector, the CCTVs are essential for the ATM surveillance.

3 FINSEC Platform Overview

FINSEC (Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures) is a Horizon 2020-funded project being developed by a consortium of 23 international partners [5]; its aim is to provide an integrated service platform for the cyber-physical security of critical financial infrastructures.

Figure 3 shows a simplified version of the FINSEC architecture [6], highlighting the main components related with TLSAssistant, which can be seen as a three-layered set of components with different roles:

- **Presentation Layer** composed by a Dashboard and any External Services with access to the APIs. In particular, the Dashboard is an interactive interface developed within the FINSEC project capable of showing the overall security status of the infrastructure;
- **Service Layer** contains the services integrated in the FINSEC platform. The services can be either called on-demand (like TLSAssistant) or have an ongoing monitoring activity (e.g., the Risk Assessment Engine);
- **Data Layer** hosts a collection of security policies, vulnerabilities (imported CVE), system logs and other intel. It is where all the knowledge is stored and retrieved by upper layer entities. Its content is written in FINSTIX, a proprietary extension of the STIX language (see Sect. 4.1 and [7]).

From a developer’s perspective, the FINSEC platform has a microservice architecture. This means that each integrated service is completely virtualized, independent from the others and must therefore handle its own dependencies and deployment. In FINSEC, this has been achieved by using the containerization technology provided by Docker. Thanks to the high level of independence, this approach aims to simplify both the overall management and each maintainer’s work on condition that they handle their containers as if they were partially cloud-deployed. This because, while each service can maintain its own private state, it has to store all the relevant information in the shared database and handle its own access queue.

The communication among containers is managed through the use of REST APIs that each maintainer must provide. The set of APIs exposed by the services is called SECaaS API while the one provided to access the Data Layer is called Data Access API.
4 Integration with FINSEC Platform

Having been developed outside the FINSEC context, TLSAssistant is the result of design choices driven by the use-case of a system administrator wanting to check his own TLS deployment by downloading TLSAssistant and being able to use it even when offline. This specific use case, notably different from the FINSEC platform, led to (i) the creation of an internal database containing the vulnerability mitigations and other information that help address the issues, (ii) the ability to run a single analysis at a time, and (iii) the generation of a human-readable report.

To maintain TLSAssistant’s autonomy while being able to satisfy all the FINSEC requirements, the integration was divided in three parts:

1. extension of TLSAssistant’s output capabilities to allow a STIX-generated report option;
2. creation of a Connector able to translate TLSAssistant’s STIX output in FIN-STIX and to link our tool with the FINSEC’s Data Layer, avoiding data redundancy and to maintain consistency across multiple analysis;
3. creation of a set of REST APIs and of a queue manager to allow concurrent requests to be served sequentially.

4.1 STIX Output In TLSAssistant

Sharing intelligence with automated services required producing structured data that are consumable by other services through a persistent data store. For this reason, from version v1.2, TLSAssistant is able to export the analysis result in STIX (Structured Threat Information eXpression) [18], a language used to share cyber threat intelligence (CTI) that can be represented with objects and their descriptive relationships. The STIX Domain Objects (SDOs) and STIX Relationship Objects (SROs) are visually summarized in Fig. 4.

An implementation of TLSAssistant with STIX version 2.0 is available on GitHub [25]. To clarify how the TLSAssistant report has been modified to map...
the vulnerability and mitigation output to STIX objects and relationships, Fig. 5 shows an example for the Bar Mitzvah attack. In general, after every scan and for each discovered vulnerability, TLSAssistant generates a JSON file containing the following entries:

- **vulnerability** is an SDO that indicates a weakness that can be used by an attacker to compromise a system. The CVE ID [14] that provides a common name for known vulnerabilities is usually present in the `external_references` property. In TLSAssistant, the vulnerability SDO contains the name and description of the detected vulnerability.

- **course of action** is an SDO used to give a recommendation on the actions that might be taken in response to a CTI. It describes technical responses (e.g., patches) or higher level actions (e.g., policy changes). In TLSAssistant, the course of action SDO contains the textual description of the mitigation (in the `description` field) and the actionable mitigation in the form of code snippets (in the `x_actions` custom field).

- **relationship** is an SRO used to link together two SDOs or STIX Cyber-observable Objects (SCOs) in order to describe how they are related to each other. STIX defines many relationship types, e.g. `uses`, `targets`, `mitigates`. **Fig. 4.** A simplified SDO ecosystem with its possible relationships.
In TLSAssistant, a course of action is linked with a vulnerability through the mitigates relationship type.

observed data is an SDO that contains information about entities (e.g., files and systems) using the SCOs to provide supporting context. It is not an intelligence assertion, it is simply the raw information without any context for what it means. In TLSAssistant, the observed data SDO contains info about the asset observed (e.g., the URL analyzed and the timestamp of the scan).

sighting is an SRO that denotes the belief that something in CTI was seen. In TLSAssistant, the sighting SRO links the vulnerability SDO and the observed data SDO through the observed and sighting of relationship types.

4.2 FINSEC Connector

The Connector is a Python script written to integrate TLSAssistant in the FINSEC platform. Figure 4 shows its two modes of operation: on_deployment and on_scan.

on_deployment. The script is called once during the container’s deployment phase. To avoid data inconsistencies or duplicates, once invoked it checks if a
deployment had already occurred. In this mode, the connector has the role of linking TLSAssistant’s intelligence to the Data Layer content. In particular:

1. it exports TLSAssistant’s internal database. This will create a set of STIX bundles (see Sect. 4.1) containing three objects: a vulnerability, a course of action, and a relationship of type mitigates;
2. using the exported vulnerabilities, it retrieves their IDs from the Data Layer then edits all the relationship objects. By doing this, each course of action will be linked with the proper object within the shared database. Two edge cases can occur:
   - if a single course of action is able to mitigate more than one vulnerability, the connector will create an “aggregated” vulnerability object and link it to the mitigation (to avoid SRO redundancy);
   - if a vulnerability extracted from TLSAssistant does not have a CVE (hence the Data Layer will not contain its object), a new vulnerability object will be created and uploaded;
3. it extends the structure of each course of action by adding FINSTIX properties (see Sect. 3). These (e.g., ‘x_subtype’ = ‘to_dashboard’) are used to extend the STIX language and manage the integration with the Dashboard;
4. lists all the created and linked vulnerabilities in a file stored locally (this is the file whose existence will be checked on startup) and uploads all course of action and mitigates objects.

**on_scan.** The script is called after every completed scan. Once started, the connector retrieves the JSON files generated by TLSAssistant (see Sect. 4.1), and performs the following operations (for each file):

1. extracts the sighting, the observed data (see Fig. 5) and the name of the detected vulnerability;
2. matches the name of the vulnerability and links the sighting to its ID in the Data Layer (value retrieved or generated during the deployment);
3. tags each sighting object with a custom scan_id field so that the initiating service could retrieve the results;
4. finally, it uploads both the sighting and the observed data to the Data Layer, allowing the Dashboard, the initiating service, and any other services to retrieve the results of the scan, now available within the shared database.

### 4.3 API and Queue Handling

Each TLSAssistant scan can take anywhere between 1 and 5 min, and an installation of TLSAssistant was never designed to handle concurrent requests. Making the service available therefore required not only the definition of an API to initiate scans, but also a message queue for requests to be passed to worker threads, and a data store for states and results to be queried.

Figure 6 shows a component diagram of TLSAssistant integrated in the FINSEC platform, with our contribution (colored) composed of the following:
– **TLSAssistant** + **Connector** to analyze the deployments and interact with the *Data Layer* (see Sects. 2.2 and 4.2);
– a set of Service APIs exposed using Flask and leveraging the Celery task queue;
– rabbitmq broker for scan requests;
– redis to store scan states.

In detail, our service expose the following Service APIs:

**POST /scan** a JSON object with a `url` key on which to initiate a vulnerability scan; returns a `scan_id` - a UUID as per RFC 4122.

**GET /state/{scan_id}** returns the current scan state as reported by Celery (e.g. pending, success and failure).

**GET /result/{scan_id}** retrieves all sighting for the given `scan_id`, and all associated objects such as vulnerability and course of action, from the *Data Layer*.

Finally, enhancing the functionality of other services to achieve a synergy beyond their consumption of the generated STIX objects required individual integration, e.g. Risk Assessment - see Sect. 5.

## 5 Integration with Risk Assessment

As discussed in the introduction, one of the most important advantages in integrating a tool like **TLSAssistant** in the FINSEC platform is to enhance the functionality of other services. Particularly relevant to any security platform is to
provide a comprehensive evaluation of the security risks of an infrastructure or system. We thus consider how TLSAssistant can contribute information to refine the calculation of the Risk Assessment Engine (RAE) available in the FINSEC platform in order to support a continuous risk evaluation process.

The RAE is designed to support a continuous monitoring of assets; this allows a Security Officer at a financial institution to maintain a list, for instance, of TLS servers in their infrastructure, including but not limited to Online Banking or PSD2 API servers, and have them regularly scanned for vulnerabilities. While changes in server configurations should only occur infrequently and as a consequence of manual intervention by a system administrator, a regular scan also mitigates against undetected malicious changes, unintended consequences e.g. of upgrades to dependencies, or automated changes. The regular scans may return observations of vulnerabilities that trigger a re-evaluation of the current risk level. If the vulnerabilities are tied to a specific CVE, this can be cross-referenced with those potentially present on the asset by manufacturer and model, and its impact can be assessed based on CVSS scores.

The RAE integrated in the FINSEC platform [8] takes an approach based on graphical risk modeling described in [27]. To summarize, this entails the creation of a CORAS model, based on an understanding of the overall risk pattern to be modeled, and the definition of risk assessment algorithms - Bayesian networks in R and decision diagrams in DEXi - for an automated quantitative and qualitative risk assessment.

The RAE has been adapted to consume STIX sighting objects to trigger re-evaluation of risk models. As noted in Sect. 4.1, TLSAssistant produces sighting objects linked to cyber observables; the RAE however measures risks associated with an x-asset object, developed specifically for the FINSTIX data model [7]. This object links specific cyber-physical entities, e.g. servers, with their cyber-physical address (e.g. LAN IP) and/or physical location, as well as parameters relevant to risk assessment. A synergy between our vulnerability scanning service and the Risk Assessment Engine can be achieved by (a) linking each scan of an address with a known x-asset, on the part of the scanning service, and (b) adding sighting of the relevant vulnerability objects to the RAE risk models.

The following modifications were made specifically to integrate with the RAE.

x-asset objects in the Data Layer had a url property added
scan/ API endpoint calls can be made with an x-asset-id JSON element; the API will retrieve the corresponding object from the Data Layer and read its url property.
x-asset-id provided to the API is added to the x_asset_refs[] property in sighting objects produced by TLSAssistant.
CORAS risk models were updated to include some of the specific CVE that TLSAssistant can produce sighting of. A simplified model of how sighting of vulnerability may be represented in CORAS is shown in Fig. 7.
6 Conclusions and Future Work

Combining different security services and merging their results in a coherent view presents some interesting challenges and opportunities. In this paper, we present how we addressed them in the integration in the FINSEC platform of an open-source tool (TLSAssistant) proposed to assist system administrators to deploy resilient instances of the TLS protocol. Firstly, for sharing TLSAssistant’s output with FINSEC services, we extended our tool with STIX 2.0, a language used to share structured cyber threat intelligence data. Secondly, as TLSAssistant was never designed to handle concurrent requests, we provided – together with the APIs to initiate scans and retrieve the results – an API to request the state of the scan and a data store for states and results to be queried. Finally, we showed how the TLSAssistant service integrated in the FINSEC platform can be easily modified to support the integration with other FINSEC services (e.g., the risk assessment engine).

As future work, we plan to further improve the TLSAssistant integration (e.g., by managing the edge cases) and the orchestration of TLSAssistant requests (e.g., by adding a scan scheduling to manage both on-demand and scans repeated on a regular basis).

Acknowledgments. This work has been supported by the research project “Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures” (FINSEC), which receives funding from the European Union’s Horizon 2020 Research and Innovation Programme under Grant agreement 786727.
References

1. Berlin Group: NextGenPSD2 Access to Account Interoperability Framework - Implementation Guidelines V1.3.4. https://www.berlin-group.org/nextgenpsd2-downloads
2. Böck, H., Somorovsky, J., Young, C.: Return of Bleichenbacher’s oracle threat (ROBOT). In: 27th USENIX Security Symposium (USENIX Security 18), pp. 817–849 (2018)
3. European Parliament: Directive (EU) 2015/2366 of the European Parliament and of the Council on payment services in the internal market, amending Directives 2002/56/EC, 2009/110/EC and 2013/36/EU and Regulation (EU) No 1093/2010, and repealing Directive 2007/64/EC. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32015L2366&from=EN
4. Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., Smith, M.: Why eve and mallory love android: an analysis of android SSL (in)security. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 50–61 (2012). https://doi.org/10.1145/2382196.2382205
5. FINSEC: Integrated Framework for Predictive and Collaborative Security of Financial Infrastructures. https://www.finsec-project.eu/
6. FINSEC D2.5: FINSEC Reference Architecture II (October 2019). https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5ce3a941d&appId=PPGMS
7. FINSEC D3.9: Finance Sector Security Knowledge Base I (October 2019), https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5ce8e14437&appId=PPGMS, due to be updated in Deliverable D3.10 in 2021
8. FINSEC D4.5: Risk Assessment Engine for Critical Infrastructures in the Financial Sector II (March 2020). https://ec.europa.eu/research/participants/documents/downloadPublic?documentIds=080166e5ccabbc21&appId=PPGMS, due to be updated in Deliverable D4.6 in 2021
9. IETF: The Transport Layer Security (TLS) Protocol: Version 1.3. https://tools.ietf.org/pdf/rfc8446.pdf
10. IMPERVA: Attacking SSL when using RC4. https://www.imperva.com/docs/HII_Attacking_SSL_when_using_RC4.pdf
11. Kario, H.: tlsfuzzer: SSL and TLS protocol test suite and fuzzer. https://github.com/tomato42/tlsfuzzer
12. Maddison, J.: Defining the security platform. https://www.csoonline.com/article/3527843/defining-the-security-platform.html
13. Manfredi, S., Ranise, S., Sciarretta, G.: Lost in TLS? no more! assisted deployment of secure TLS configurations. In: 33th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec), pp. 201–220 (2019). https://doi.org/10.1007/978-3-030-22479-0_11
14. MITRE: Common Vulnerabilities and Exposures. https://cve.mitre.org/
15. Network Working Group: The TLS Protocol: Version 1.0. https://tools.ietf.org/pdf/rfc2246.pdf
16. Network Working Group: The Transport Layer Security (TLS) Protocol: Version 1.1. https://tools.ietf.org/pdf/rfc4346.pdf
17. Network Working Group: The Transport Layer Security (TLS) Protocol: Version 1.2. https://tools.ietf.org/pdf/rfc5246.pdf
18. OASIS Open: STIX - A structured language for cyber threat intelligence. https://oasis-open.github.io/cti-documentation/stix/intro
19. OpenID Foundation: Financial-grade API (FAPI). https://fapi.openid.net/
20. PCI Security Standards Council: Requirements and security assessment procedures v3-2-1. https://www.pcisecuritystandards.org/document_library (2018)
21. PRETA Open Banking Europe: Security and Identification Standards for APIs & Communications. https://www.openbanking-europe.eu/media/1398/oasis-obe-api-identification-and-security-standards-for-apis-and-communications.pdf
22. Qualys: SSL Pulse. https://www.ssllabs.com/ssl-pulse/
23. Ronen, E., Gillham, R., Genkin, D., Shamir, A., Wong, D., Yarom, Y.: The 9 lives of Bleichenbacher’s CAT: new cache ATtacks on TLS implementations. IEEE Symposium on Security and Privacy, SP 2019, 435–452 (2019). https://doi.org/10.1109/SP.2019.00062
24. Samarasinghe, N., Mannan, M.: Short paper: TLS ecosystems in networked devices vs. web servers. In: Financial Cryptography and Data Security - 21st International Conference, FC 2017, pp. 533–541 (2017). https://doi.org/10.1007/978-3-319-70972-7_30
25. Security & Trust Research Unit: TLSAssistant. https://github.com/stfbk/tlsassistant
26. Sheffer, Y., Holz, R., Saint-Andre, P.: Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS). https://tools.ietf.org/html/rfc7457.pdf
27. Černivec, A., Erdogan, G., Gonzalez, A., Refsdal, A., Alvarez Romero, A.: Employing graphical risk models to facilitate cyber-risk monitoring - the WISER approach. In: Graphical Models for Security (GraMSec) 2017. LNCS, vol. 10744, pp. 127–146 (2018). https://doi.org/10.1007/978-3-319-74860-3_10
28. Waked, L., Mannan, M., Youssef, A.M.: The sorry state of TLS security in enterprise interception appliances. CoRR abs/1809.08729 (2018). http://arxiv.org/abs/1809.08729
29. Wetter, D.: /bin/bash based SSL/TLS tester: testssl.sh. https://testssl.sh