Limitations of Wrapping Protocols and TLS Channel Bindings: Formal-Methods Analysis of the Session Binding Proxy Protocol

Enis Golaszewski[1], Edward Zieglar[2], Alan T. Sherman[1], Kirellos Abou Elsaad[1] and Jonathan Fuchs[1]

[1] Cyber Defense Lab, University of Maryland, Baltimore County (UMBC), Baltimore, USA
[2] National Security Agency, Fort George G. Meade, USA

Abstract.
We present the first formal-methods analysis of the *Session Binding Proxy (SBP)* protocol, which protects a vulnerable system by wrapping it and introducing a reverse proxy between the system and its clients. SBP mitigates thefts of authentication cookies by cryptographically binding the authentication cookie—issued by the server to the client—to an underlying *Transport Layer Security (TLS)* channel using the channel's master secret and a secret key known only by the proxy. An adversary who steals a bound cookie cannot reuse this cookie to create malicious requests on a separate connection because the cookie's channel binding will not match the adversary's channel. SBP seeks to achieve this goal without modifications to the client or the server software, rendering the client and server *"oblivious protocol participants"* that are not aware of the SBP session.
Our analysis verifies that the original SBP design mitigates cookie stealing under the client's cryptographic assumptions but fails to authenticate the client to the proxy. Resulting from two issues, the proxy has no assurance that it shares a session context with a legitimate client: SBP assumes an older flawed version of TLS (1.2), and SBP relies on legacy server usernames and passwords to authenticate clients. Due to these issues, there is no guarantee of cookie-stealing resistance from the proxy's cryptographic perspective.
Using the *Cryptographic Protocol Shapes Analyzer (CPSA)*, we model and analyze the original SBP and three variations in the Dolev-Yao network intruder model. Our models differ in the version of TLS they use: 1.2 (original SBP), 1.2 with mutual authentication, 1.3, and *1.3 with mutual authentication (mTLS-1.3)*. For comparison, we also analyze a model of the baseline scenario without SBP. We separately analyze each of our SBP models from two perspectives: client and proxy. In each SBP model, the client has assurance that the cookie is valid only for the client's legitimate session. Only in mTLS-1.3 does the proxy have assurance that it communicates with a legitimate client and that the client's cookie is valid. We formalize these results by stating and proving, or disproving, security goals for each model.
SBP is useful because it provides a practical solution to the important challenge of protecting flawed legacy systems that cannot be patched. Our analysis of this obscure protocol sheds insight into the properties necessary for wrapper protocols to resist a Dolev-Yao adversary. When engineering wrapper protocols, designers must carefully consider authentication, freshness, and requirements of cryptographic bindings such as channel bindings. Our work exposes strengths and limitations of wrapper protocols and TLS channel bindings.

E-mail: golaszewski@umbc.edu (Enis Golaszewski), evziegl@uwe.nsa.gov (Edward Zieglar), sherman@umbc.edu (Alan T. Sherman), abou3@umbc.edu (Kirellos Abou Elsaad), jfuchs2@umbc.edu@umbc.edu (Jonathan Fuchs)

1 Introduction

Flawed legacy systems are a security professional's nightmare: they are challenging to patch against known vulnerabilities, store large accumulations of sensitive data, and often contribute to spectacular data breaches [SN20]. In the 2015 attack on the United States *Office of Personnel Management (OPM)*, which leaked over 22 million sensitive records to geopolitical adversaries, attackers exfiltrated data from decades-old computer systems incapable of encryption [PT17]. To protect legacy systems, organizations often attempt to isolate such systems by using *Virtual Local area Networks (VLANs)*, firewalls, air gaps, and reverse proxies. In spite of these attempts, poorly designed solutions continue to enable existing or new attacks, while instilling a false sense of security.

Legacy systems often communicate using outdated cryptographic protocols or legacy protocols, with known weaknesses. A prevalent approach for hardening legacy protocols is to bind them to modern protocols, such as *Transport Layer Security (TLS)*, and to rely on this binding to mitigate known vulnerabilities. Cryptographic channel binding (see Section 3.2), a common binding technique, binds the cryptographic values of the legacy protocol to a cryptographic channel established by a modern protocol (e.g., TLS). Because it is difficult to develop sound cryptographic protocols, due diligence requires that we analyze cryptographic bindings using formal methods.

We present a formal-methods analysis of the 2013 *Session Binding Proxy (SBP)* protocol by Burgers, Verdult, and Eekelen [BVvE13], originally designed to mitigate a known session-hijacking vulnerability with their university's Blackboard Learn software [vEMHV13]. SBP is a cryptographic protocol that wraps a vulnerable web application server and introduces a *reverse proxy*. SBP binds the HTTP *session authentication cookie*—a token with which a client identifies themselves to the server—to an underlying TLS channel between the client and the proxy. The session binding reverse proxy receives requests on behalf of the server and manages the cookie binding. A design goal of SBP is to avoid modifying the client or the server: neither the client nor the server are aware that they participate in a session of SBP with the proxy.

To analyze SBP, we model a baseline scenario without SBP (pre-SBP), the original SBP, and three custom variations in the strand space model [THG99] using the *Cryptographic Protocol Shapes Analyzer (CPSA)* [GLRR]. Throughout, we focus on SBP's goal of preventing cookie-stealing attacks. Our model of the original SBP incorporates and binds to TLS-1.2, consistent with the SBP authors' written description and their prototype implementation. Each of our custom variations binds to alternate versions of TLS: *mutual TLS (mTLS)*-1.2, TLS-1.3, and mTLS-1.3 Using our models, we explore how variations between underlying TLS channels affect SBP's cookie-stealing resistance and compare these to pre-SBP.

Using CPSA, we formalize security goals of the SBP design by extracting authentication goals to define an SBP session context. From each of the protocol's distinct cryptographic perspectives (client, proxy), we prove *context agreement*, in which legitimate parties agree on the session context after executing the protocol, via an exhaustive proof or a counterexample. To our knowledge, we are the first to perform a formal-methods analysis of the SBP protocol.

From the cryptographic perspectives of the client and proxy, our analysis identifies the properties of the underlying communication channel that SBP requires to function within the *Dolev-Yao (DY)* network intruder model [DY83]. The client and the proxy must mutually authenticate each other, and the channel binding must incorporate fresh values from each communicant. From the proxy's perspective, the original SBP, mTLS-1.2, and TLS-1.3. models fail to authenticate mutually: the proxy cannot determine if they are communicating with a legitimate client or the network adversary. Within a DY network, traditional passwords are an inadequate authentication mechanism because clients may inadvertently transmit these to penetrator strands (see Section 13). Additionally, the proxy

in the original SBP and mTLS-1.2 models has no assurance that the client contributes a fresh pre-master secret to the protocol—consequently, the proxy may bind the cookie to a compromised TLS channel.

Unlike in our SBP TLS-1.2 and TLS-1.3 models, in our SBP mTLS-1.3 model, the proxy authenticates the client using a client certificate; the proxy is guaranteed freshness of the TLS master secret; and the protocol does not leak this freshness—preventing session hijacking. In this variation, the client receives a cookie bound to the fresh TLS master secret. Of the models we analyze, only this model guarantees that both the client and the proxy perspectives agree on an SBP session context, including the identities of the user, the server, and the cookie. The client certificate, however, does not bind to a username or password, enabling the client to transmit stolen credentials to the proxy on the mutually authenticated channel. To mitigate this issue, we recommend eliminating password-based authentication in favor of user certificates or other mechanisms.

From our analysis, we identify several vulnerabilities resulting from the design of the original SBP protocol: (1) The proxy cannot authenticate the client because the client does not produce a certificate; the client may transmit stolen credentials; and the protocol leaks freshness of the TLS pre-master secret. (2) The client may communicate with a corrupt proxy that forwards the client's credentials and requests in another session of SBP. (3) The client cannot verify the cryptographic binding that the proxy applies to the cookie. As we show in Section 9.2, an adversary can exploit these vulnerabilities to launch a *"forwarding attack."*

To mitigate SBP's vulnerabilities, we make several recommendations. SBP must require that the server embed the proxy to merge into a single network entity. Clients must produce certificates to authenticate via mTLS-1.3, and the certificates must bind to specific usernames and passwords to mitigate stolen passwords. The proxy must provide the client with some means by which to verify the binding of the authentication cookie (see Section 13). We also point out how SBP's design prevents meaningful improvements without disruptively modifying the server and client (see Section 10).

This work evolves a preliminary formal-methods analysis of SBP by Elsaad [AE22], who verified that the original SBP protocol resists protocol interactions from the client's perspective. Additionally, as we present in Section 9.2, Elsaad identified and implemented a "tail-gating attack" against SBP, which (beyond the DY model) assumes that the adversary can execute code in the client's browser. Our expanded study analyzes models of the original SBP and three variations using alternate TLS versions and compares them with a pre-SBP model. We also identify issues with TLS channel binding and passwords, and make recommendations for designing and deploying protocols like SBP.

Our primary contributions are: (1) We present a formal-methods analysis of the SBP protocol, in which we state and prove security goals from the server and proxy perspectives for a model of the original SBP, three variations, and a baseline model without SBP. (2) We interpret our formal analysis in terms of vulnerabilities, attacks, and risks; and (3) we recommend principles and improvements for SBP and similar wrapper protocols.

In the remainder of this paper, we introduce SBP and wrapping protocols that perform channel binding, review relevant background, explain the SBP protocol, state our adversarial model, present our CPSA model for each variation of SBP, state security goals in strand space for each model, prove or disprove the security goals using CPSA, point out potential vulnerabilities, attacks, and risks, make recommendations, summarize previous work, and discuss issues and open problems raised by our work.

Source code for our CPSA models are available on GitHub [UMB23].

2 Introducing SBP, Wrapping Protocols, and TLS Channel Bindings

Wrapping protocols such as SBP, which encapsulates an older, flawed protocol, exist to solve a practical problem facing many organizations: hardening legacy systems against adversaries who target these systems with known exploits. Such adversaries exploit known vulnerabilities in these systems that can be impossible for organizations to patch. Systems may become impossible to patch because vendors discontinue support, developers of custom software become unavailable, and patches break the system's functionality. Although it may be tempting to eliminate the legacy system from an organization's network, doing so may be prohibitively expensive and disruptive. Solutions such as SBP attempt to wrap and isolate legacy systems to keep them available while mitigating known attacks.

TLS channel binding is the key idea of SBP, which seeks to improve cookie-based authentication in HTTP by applying channel binding to the cookie. Many protocols assume authenticated communication channels negotiated by protocols such as TLS, which is widely available, supported, and ubiquitous with web browsing. To bind values in a new protocol to an underlying TLS channel, protocol designers apply channel bindings: they encrypt or hash their sensitive values together with secret values originating from the TLS channel. The purpose of such bindings is to enable protocol participants to detect protocol interactions in which an adversary attempts to transfer information bound to one TLS channel to a session taking place on a different TLS channel. We explore the ability of SBP's channel bindings to mitigate cookie-based hijacking.

Wrapper protocols like SBP introduce the problematic notion of *oblivious protocol participants* (see Figure 1), that participate in cryptographic protocols without knowledge that they are participating. Often, these participants are flawed systems that are not possible to patch. To enable such systems to function in a network without presenting soft targets for adversaries, wrapper protocols fool the participants into participating in stronger protocols that mitigate known flaws. The existence of protocols such as SBP illustrates a need for such protocols and their oblivious participants.

Oblivious protocol participants introduce new dangers, particularly in the DY model, because they are not aware of the protocols or sessions in which they communicate. Cryptographic bindings (see Section 3.2) are vital primitives for preventing man-in-the-middle attacks but are impossible for oblivious participants to apply or verify.

3 Background

3.1 Session Hijacking

In session hijacking, an adversary presents a legitimate user's session credentials as their own, potentially enabling malicious transactions. Examples include *cross-site scripting (XSS)*, in which the adversary injects code into the client's browser to steal a session authentication token [RTFB20], and malicious browser plugins [KFS+22, JDG+15, KGC+14], which can modify web pages in the client's browser or steal sensitive information, such as an HTTP cookie. Existing mitigations include HTTP-only cookies [ZE10]. SBP seeks to eliminate cookie-based session hijacking by binding the user's cookie to an underlying TLS session.

3.2 Cryptographic Binding

Cryptographic binding is vital for preventing a DY adversary from creating adverse protocol interactions [KSW97]. It applies cryptographic primitives, such as encryption, digital signature, or hashing, to associate session context (e.g., identities and nonces) with a message from a specific protocol session. Cryptographic binding may or may not involve

secrets. It is important that an adversary cannot undo bindings and that recipients be able to verify them. As early as 1996, Abadi and Needam's [AN96] informal guidelines on designing sound cryptographic protocols stated a need to bind to session contexts cryptographically. An example of cryptographic binding is digital certificates, which associate public keys with identities via signature.

Protocols that do not bind appropriately are vulnerable to protocol interactions. The well-known 1995 attack by Gavin Lowe [Low95] on the *Needham-Schroeder (NS)* [NS78] protocol exploits a lack of binding between the responder's nonce and the responder's identity. Binding failures continue to appear in modern protocols such as FIDO UAF [GSZ23], illustrating the need for formal-methods tools in protocol design. We discuss requirements for effective cryptographic bindings in Section 13.

3.3 Formal-Methods Protocol Analysis

Formal-methods protocol analysis involves expressing a protocol in a formal, mathematical model, stating theorems that reflect the protocol's desired security properties, and proving those theorems. Often, this process requires the assistance of specialized theorem-proving tools. Notable existing tools for protocol analysis include ProVerif [Bla13], Tamarin Prover [MSCB13], Maude-NPA [EMM07], and CPSA [LRGR16]. It is also possible to carry out security analysis using more general high-order logic theorem provers such as Isabelle [Pau98]. Unlike most theorem provers, CPSA is capable of discovering protocol security goals given a set of assumptions and a partial execution. In this sense, CPSA is a "model-finding tool." Like other theorem provers, CPSA is also capable of verifying stated theorems within the model. For our analysis, we use CPSA because we are familiar with the tool, have access to experts, and find the tool's model-finding properties useful for identifying protocol security goals and protocol interactions.

3.4 Strand Spaces

Strand spaces [THG99] are a formalization of interactions between protocol participants on a DY network. A protocol's *strand space* comprises a set of *strands*, which express actions by legitimate parties or a network penetrator. Each strand expresses a sequence of incoming and outgoing messages via positively or negatively signed *terms*: positive terms denote messages going out into the network, and negative terms denote messages coming in from the network. A pair of strands with corresponding terms, such that one strand transmits the terms that the other strand receives, are *complementary* strands. We model the strand space of a protocol by extracting the protocol's roles from its specification, extracting complementary strands from these roles, and including penetrator strands that express the capabilities of our adversary.

Together with a set of keys known to the adversary, penetrator strands express the adversary's available actions on the network. The adversary composes these strands to construct attacks within a protocol's strand space. A standard set of penetrator strands models the following actions: (1) Emit or extract arbitrary messages. (2) Replay messages. (3) Concatenate message or separate messages into components. (4) Using knowledge of a secret key, encrypt or decrypt messages. When necessary, one can include additional penetrator strands that express additional behaviors.

Interactions between strands occur in subsets of strand spaces called *bundles*: a bundle consists of a portion of legitimate party strands and penetrator strands from a strand space and captures causal relationships between the terms that these strands transmit and receive. Strands within bundles follow three rules: First, a strand cannot send and receive a message concurrently. Second, a strand must receive a message from a unique node on another strand that emits that message. Third, if a strand emits a message, any strand expecting such a message may receive it. The same strand can appear multiple times in

a bundle. Bundles express unique execution sequences within a protocol's strand space, including sequences that describe attacks by an adversary.

Because proving properties of strand spaces is subtle, time consuming, and laborious, we rely on CPSA to prove theorems for us automatically.

3.5 CPSA

CPSA [LRGR16] is an open-source tool for analyzing cryptographic protocols within the strand-space model. While capable of proving theorems, CPSA distinguishes itself as a model-finder: given an input model—which comprises strands consisting of roles, messages, variables, and a set of initial assumptions—when executing to completion, CPSA identifies all essentially different executions of the protocol within a DY network and outputs these executions as *shapes* [LRT11]. CPSA's model finding enables users to identify the strongest achieved security goal for an input model [RGL16]. Additionally, for each shape, CPSA outputs shape analysis sentences from which users can extract the security goals [Gut00, Gut14, Ram15] and corresponding proofs.

Users define CPSA models using LISP-like s-expressions that implement a custom language. In these models, which superficially resemble (but are not) executable source code, users specify one or more roles, associated variables and messages, and skeletons. *Skeletons* specify one or more initial strands and impose assumptions on the strand variables, such as "uniquely originating" a value or identifying a value, most often a secret or key, is unavailable to the adversary. When CPSA executes, the tool attempts to satisfy skeleton nodes by repeatedly applying actions available to a DY intruder in strand space theory. A *shape* is a skeleton consisting of only the strands of legitimate protocol participants, in which CPSA satisfies all nodes under the skeleton's assumptions.

Within a CPSA skeleton, we constrain variables by applying one or several origination assumptions: uniquely originating, uniquely generating, non-originating, and penetrator non-originating. *Uniquely originating* values are unknown to protocol participants and the network until a legitimate strand emits them as part of a message, enabling us to model random nonces, fresh secret keys, and other values that must be unique for each execution of a protocol. *Uniquely generating* values are unknown to protocol participants until a legitimate strand creates them. They are distinct from uniquely originating in that originating terms do not exist until they are sent in a message, while generating terms are unique on the node within a strand where they first appear. *Non-originating* values are values such as private keys, which the adversary does not know, cannot guess, and will never appear on the network in a decryptable form. *Penetrator non-originating* values include passwords, which only legitimate strands can originate, that the adversary does not know without obtaining them from some run of the protocol.

3.6 Proxies

Proxies, also known as proxy servers, are systems that pass messages on behalf of other systems, often between web clients and servers [Luo98, WKDP14]. There are two major types of proxies: *forward proxies*, which accept communication from entities and pass them to remote destinations on a network, and *reverse proxies*, which appear outwardly as servers and pass traffic to other systems.

Proxies intercept and manipulate network traffic by design and can appear indistinguishable from an adversary, making them problematic for protocol security. A rogue proxy enables an adversary to create, delete, and modify network messages to exploit flaws in network protocols. Because communicants often are unable to determine that they are communicating via a proxy, the scenario of a rogue proxy can be challenging to detect and mitigate. Within the DY model, a potentially corrupt proxy may make achieving many

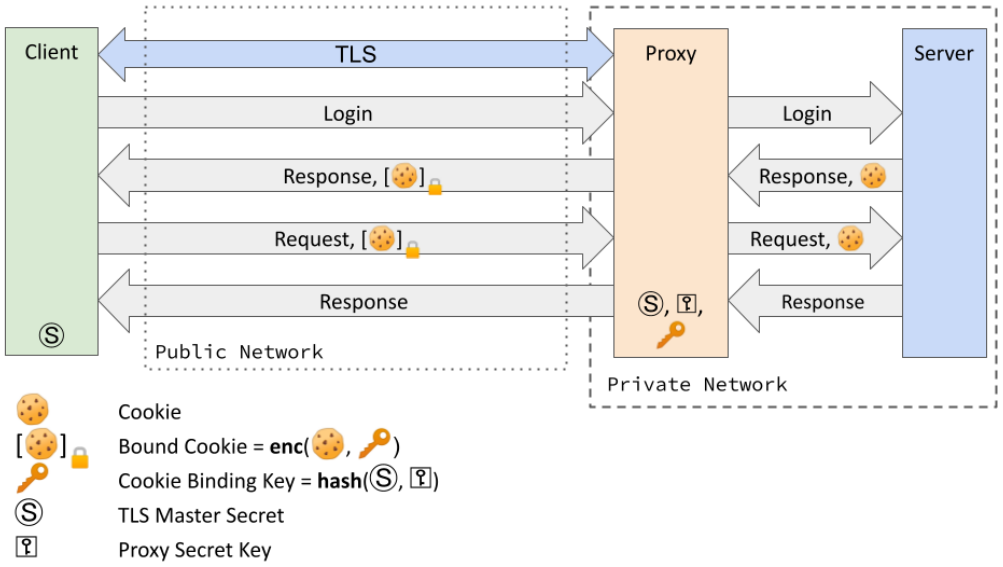

Figure 1: Architectural diagram for the SBP protocol in which the client and application server are oblivious protocol participants. A client establishes a session with the SBP (reverse) proxy over the Internet and issues further requests to this proxy. The proxy strips SBP-specific information from the client's messages before forwarding them to the application server across a private network. The server, believing the proxy to be the client, replies with responses that the proxy relays to the client.

security goals impossible. We discuss the potential vulnerabilities and attacks affecting SBP's reverse proxy in Section 9.

3.7 Transport Layer Security (TLS)

Transport Layer Security (TLS) [Res01] is a widely deployed cryptographic protocol for negotiating encrypted communication channels on insecure networks. Frequently, protocols that require an encrypted channel with integrity bind with TLS. The most common deployments of TLS assume a client-server architecture in which the server produces for the client a certificate, signed by a certificate authority, that attests the server's public key. This common deployment results in one-way authentication: the client authenticates the server, but the server does not authenticate the client. In a variation of TLS known as *mutual TLS*, or *mTLS*, both the client and the server produce certificates to authenticate each other.

Currently, two versions of TLS are in active use: TLS-1.2 and TLS-1.3. In TLS-1.2, the *channel master secret* comprises nonces sent in the clear from the client and the server, together with a client-generated *pre-master secret*. While the client and server contribute to *freshness*—uniqueness to a single protocol session—of the pre-master secret, TLS-1.2 leaks the freshness nonces, which can enable session hijacking. TLS-1.3 improves on this limitation by replacing the pre-master secret with a Diffie-Hellman exchange in which the client and the server securely contribute freshness.

4 Session Binding Proxy (SBP) Protocol

Developed in 2013 by Burgers, Verdult, and van Eekelen [BVvE13], the *Session Binding Proxy (SBP)* protocol is a custom protocol that wraps a vulnerable web server to improve resistance against cookie-based session hijacking. The protocol specifies a reverse proxy that manages communication on behalf of the server and cryptographically binds a client's authentication cookie to the master secret of the underlying TLS channel (version 1.2, in the original SBP). This binding hinders an adversary from stealing a client's cookie and presenting it as their own by enabling the proxy to invalidate cookies that fail to bind to the correct TLS channel. SBP specifies messages between three distinct roles (client, proxy, server) and comprises three separate protocol phases: key establishment, session establishment, and request handling. Figure 1 illustrates the binding of the cookie, the main idea behind SBP.

Key establishment. The client initiates this phase by completing a standard TLS handshake with the proxy. In this handshake, the proxy sends a certificate to the client, enabling the client to authenticate the proxy. Following the TLS handshake, the client and proxy each possess a master secret k, which the proxy uses to bind the session cookie in the following phase. The remaining phases communicate over this encrypted TLS channel.

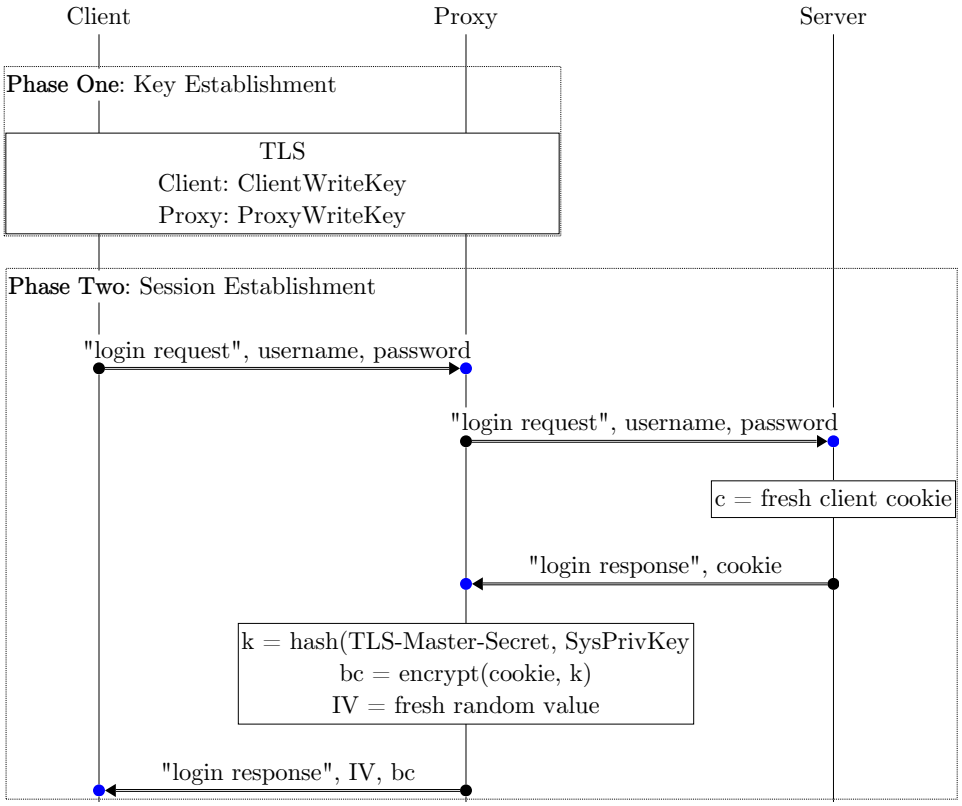

Figure 2: Idealized message sequence diagram of SBP's first two phases: key establishment and session establishment. Vertical lines correspond to protocol roles; arrows indicate sending or receiving a message; and arrow labels specify message content. The proxy passes all messages between the client and the server, binding the cookie in the second phase using the master secret of the client-proxy TLS channel and a system private key known only to the proxy.

Session establishment. To establish a session with the server, the client, server, and

proxy exchange the following messages: (1) The client transmits an authentication request, containing credentials intended for the server, to the proxy. (2) The proxy forwards the client's request to the server. (3) The server generates a session cookie, an authentication token representing the client's authenticated identity, and a response for the client, and transmits these to the proxy. (4) The proxy encrypts the session cookie using k_c, which comprises a hash of the master secret k from the previous phase, a secret system key k_p known only by the proxy, and a random initialization vector IV. The proxy transmits to the client the server's response, the IV, and an encryption of the session cookie under k_c. By so encrypting the cookie, the proxy binds the cookie to the TLS session. The client, upon receiving the final message of this phase, stores the IV and the encrypted cookie for future requests. Figure 2 illustrates the message flow of SBP's first two phases.

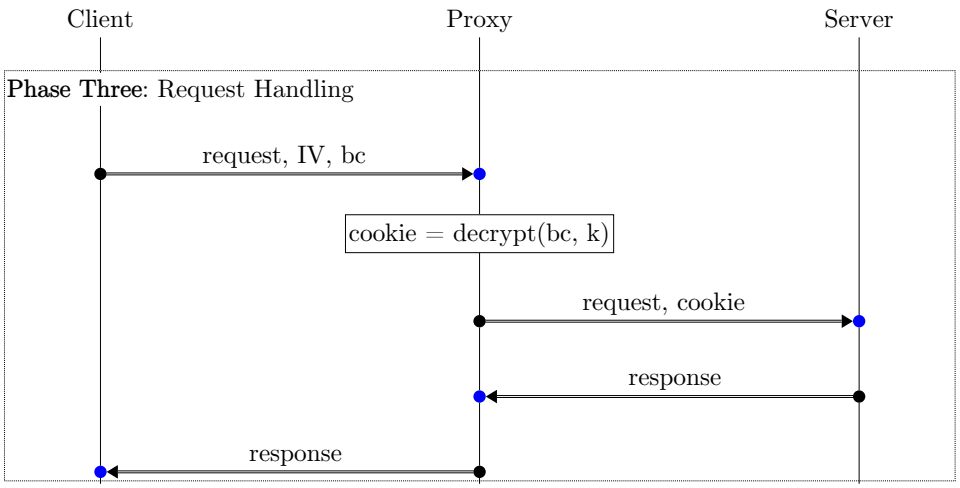

Figure 3: Idealized message sequence diagram of SBP's third phase: request handling. Using knowledge of the TLS channel with the client, the proxy decrypts the client's bound cookie *bc* and substitutes the unbound cookie in the request prior to forwarding it to the server. The server is never aware of the exchange of messages between the client and the proxy; rather, it assumes the proxy is the client and responds to the proxy appropriately. Similarly, the client is unaware of the exchanges between the proxy and the server, and the client is unaware that its cookie is a bound cookie.

Request handling. The client, now in possession of the bound session cookie, initiates further requests in this last phase: (1) The client transmits a request, the IV from the previous phase, and the bound session cookie to the proxy. (2) Using knowledge of the IV, k_c, and k_p, the proxy decrypts the client's bound cookie, replaces the encrypted cookie with the decryption of the bound cookie, and forwards the request to the server. (3) The server receives the request and the session cookie, validates the cookie, and if the cookie is correct, transmitting a response to the proxy; and (4) the proxy forwards the response to the client. Figure 3 illustrates the message flow of SBP's third phase.

The authors of SBP provide a proof-of-concept implementation that implements the proxy as an embedded component of the server, as we recommend (see Section 11). The authors, however, potentially dangerously do not require this configuration: the protocol's abstract design specifies the proxy and the server as two separate network entities that communicate over a dedicated, private channel, whose characteristics could greatly impact the security of the protocol.

5 Adversarial Model

We evaluate SBP against a DY-style adversary that hijacks an honest user's session via two primary methods: stealing a user's cookies using techniques such as XSS (see Section 3.1) and exploiting protocol interactions between two separate SBP sessions. To hijack a user's session, the adversary attempts to provide a legitimate SBP proxy with an honest user's authentication cookie.

Consistent with the DY model, the adversary has full control over all messages on the network and possesses the additional capability to steal cookies from legitimate users at will. The adversary cannot break cryptographic primitives. They must acquire session keys to compromise TLS channels or any corresponding channel bindings for any cookies they obtain. Additionally, the adversary may operate a legitimate-appearing SBP proxy with which legitimate users communicate. Within the strand space model, we formalize these capabilities as penetrator strands (see Section 3.4). To reflect the adversary's ability to steal cookies, we grant these strands the ability to originate any cookie in the protocol.

To enable TLS channel negotiation between legitimate network participants, we assume there exists an honest certificate authority on the network for which all legitimate participants possess an authentic certificate. This certificate authority produces a legitimate certificate for any network entity, including SBP proxies and users operated by the adversary. Additionally, we assume that all network participants rely on the same version of TLS, which varies for our different protocol models. Finally, we assume that the server behaves honestly and that the adversary cannot compromise messages between an SBP proxy and its corresponding server. We make this assumption because the reference implementation of SBP embeds the proxy and the server on a single network entity, and SBP will fail to achieve any security properties if this assumption is untrue.

Our adversary is willing to risk detection and criminal charges to access a legitimate user's accounts unlawfully and to engage in a range of malicious activities, such as transferring assets to steal them, collecting private information to blackmail users, or selling user credentials to criminals. Potential targets include universities (SBP's original use case), financial institutions, E-commerce services, social media, and other lucrative stores of sensitive data.

6 CPSA Models of SBP

Using CPSA's modeling language, we specify roles, variables, messages, and assumptions to model a baseline protocol (with no SBP) and four variations of the SBP protocol: *Pre-SBP*, *SBP-TLS-1.2*, *SBP-mTLS-1.2*, *SBP-TLS-1.3*, *SBP-mTLS-1.3*. The baseline pre-SBP model contrasts with SBP to illustrate how SBP helps resist our adversary. For each SBP model, a *client* communicates with a *proxy* using a version of TLS that corresponds to the protocol's name. For these models, we state security goals in Section 7 and analyze the goals in Section 8.

The pre-SBP and SBP models share several common terms that include the client's username u and password p, an authentication cookie *cookie*, and an HTTP *request* and *response*. For TLS authentication, we incorporate separate models reflecting different TLS versions into our pre-SBP and SBP models—our analysis of these models indicate that their properties are consistent with existing formal analyses of TLS. Every legitimate strand completes TLS authentication prior to sending or receiving further messages. For non-mutual TLS sessions, only server and proxy strands transmit certificates. For mutual TLS sessions, client, server, and proxy strands all transmit certificates. Following the negotiation of a TLS session, all legitimate strands in our models encrypt messages under the appropriate TLS key.

Each strand in each model corresponds to a complementary strand such that messages

in one strand mirror messages in a complementary strand. Client strand terms correspond to proxy or server strand terms. Server and proxy strand terms correspond with client strand terms. TLS authentication messages reflect these relationships between strands.

The SBP protocol enables two branches of operation for a client: the client can authenticate to the proxy, receive a bound session cookie, and make requests, or the client can recall a cookie from a previous session to make requests. We reflect this branching behavior by specifying two variations of each strand (client, proxy, server). Strands with the suffix -a authenticate using credentials then process requests. Strands with the suffix -r recall cookies from previous sessions and process requests. By including both types of strands in our strand spaces, we enable CPSA to explore behaviors resulting from strands, executing these two distinct cases, interacting with each other.

In our models, we distinguish between a client and a user. This distinction is important when introducing client certificates, as we do in SBP-mTLS-1.2 and SBP-mTLS-1.3, because client certificates serve only to authenticate the client, not the user for which the client presents credentials. Because the client certificate does not bind to a user, an authenticated client can present stolen credentials to a proxy.

6.1 Pre-SBP Model

For comparison, we begin with a pre-SBP model in which a *client* communicates with a *server* over a TLS-1.2 channel without SBP. We model four strands: client-a, client-r, server-a, and server-r. Each strand establishes a TLS session with a complementary strand and carries out its respective steps: (1) Client-a transmits u and p, receives and stores *cookie*, transmits *cookie* together with *request*, and receives *response*. (2) Client-r retrieves *cookie* from storage, transmits it with *request*, and receives *response*. (3) Server-a receives u and p, creates, transmits and stores *cookie*, receives *request* together with *cookie*, and transmits *response*. (4) Server-r receives *request* together with *cookie*, retrieves cookie from storage, and transmits *response*. Each client strand shares the client perspective, and each server strand shares the server perspective.

We also model the pre-SBP protocol over a TLS-1.3 channel and with mutual TLS. The security properties of each model are equivalent when communicating over a TLS-1.2 and a TLS-1.3 channel, so we refer to both of these models as a single model: Pre-SBP.

6.2 SBP Models

For each of our SBP models, we model four strands: client-a, client-r, proxy-a, and proxy-r. SBP strands include three additional terms over the Pre-SBP model: a system private key *ppk* for binding the cookie, the TLS channel's master secret *tls*, and a bound cookie $\mathit{bound\ cookie} = \mathit{enc}(\mathit{cookie}, \mathit{hash}(\mathit{tls}, \mathit{ppk}))$. We specify the following strands: (1) Client-a transmits u and p, receives and stores *bound cookie*, transmits the *bound cookie* with *request*, and receives *response*. (2) Client-r retrieves *bound cookie* from storage, transmits it together with *request*, and receive *response*. (3) Proxy-a receives u and p, creates and stores *cookie*, transmits *bound cookie*, receives *request* and the *bound cookie*, and transmits *response*. (4) Proxy-r retrieves *cookie*, receives *request* together with *bound cookie*, and transmits *response*. Client strands assume the client perspective, and proxy strands assume the proxy perspective.

7 Security Goals in Strand Spaces

We formalize SBP's security properties within the strand space model by specifying *context agreement* security goals: following a run of SBP, instances of a client and a proxy strand must agree on a session context. Failure to agree on context indicates the possibility of

protocol interaction, which may lead to vulnerabilities and subsequent attacks on our SBP models. For a protocol to be correct, all legitimate parties that execute a protocol must agree on the resulting session context. In our subsequent analysis, we illustrate and discuss counterexamples to goals.

A session context consists of authentication goals on which complementary pairs of role strands must agree. These facts express equivalences of values such as communicant identities, key material, nonces, or cookies. In SBP, a client and a proxy must agree on the username, the server's identifier, and a relationship between an unbound cookie and a bound cookie: for any cookie, the client must have assurance that the cookie binds to the underlying TLS channel between itself and the proxy. For the pre-SBP model, we rely on a simpler context in which a client and a server agree on a cookie, the username, and the server's identifier. Using CPSA, we iteratively arrived at minimal session contexts for the pre-SBP and SBP models, which we describe in Context 1 and Context 2, respectively.

SBP was designed to protect protocols that use cookies as authentication tokens. For such protocols, a user authenticates with the server, and the server issues a cookie to the authenticated user. The cookie is used as the user's proof of authenticated identity to that server in future requests, to eliminate the need to perform an authentication with each request. The inclusion of a valid cookie in any request allows the server to treat the request as coming from the authenticated user the cookie represents. Because cookies provide a binding between a server and a user, the minimal context that the server and client must agree upon for an authenticated exchange is the cookie, server, and user.

Session Context 1 (Pre-SBP)**.**
Let $\mathcal{C}$ and $\mathcal{S}$ be instances of complementary client strands and server strands in a pre-SBP strand space. For any pair $(\mathcal{C}, \mathcal{S})$, the following facts hold: $\mathcal{C}$ and $\mathcal{S}$ agree on the username u, the server identifier s, and the cookie.

Session Context 2 (SBP)**.**
Let $\mathcal{C}$ and $\mathcal{P}$ be instances of complementary client strands and proxy strands in a SBP strand space. For any pair $(\mathcal{C}, \mathcal{P})$, the following facts hold: $\mathcal{C}$ and $\mathcal{P}$ agree on the username u, the proxy identifier p, and for any cookie c that $\mathcal{P}$ originates, $\mathcal{C}$ must hold a corresponding bound cookie that incorporates c.

We specify three sets of origination assumptions for our distinct roles: client, proxy, and the pre-SBP model's server. Origination assumptions express a role's cryptographic perspective by establishing assumptions about values in the protocol, such as the secrecy of private keys and freshness of session values. Resulting from a disparity in origination assumptions, security goals that hold for one set of assumptions may fail to hold for another set of assumptions. Often, such a discrepancy suggests a flaw in a protocol. Below, we provide definitions for each of our assumption sets.

Assumption 1 (Client Assumptions)**.**
For any initial client strand $\mathcal{C}$ in a bundle, we make the following assumptions:
(1) Private keys of legitimate parties are unknown (non-originating) to the adversary.
(2) $\mathcal{C}$ generates (uniquely originates) a fresh client-random nonce for TLS, and [for TLS-1.2] a fresh pre-master secret or [for TLS-1.3] a fresh Diffie-Hellman value.
(3) [SBP models only] The proxy system key is not available to the adversary.

Assumption 2 (Proxy Assumptions)**.**
For any initial proxy strand $\mathcal{P}$ in a bundle, we make the following assumptions:
(1) Private keys of legitimate parties are unknown to the adversary.
(2) $\mathcal{P}$ generates a fresh proxy-random nonce for TLS, and [for TLS-1.3] a fresh Diffie-Hellman value.

Assumption 3 (Server Assumptions)**.**
For any initial server strand $\mathcal{S}$ in a bundle, we make the following assumptions:
(1) Private keys of legitimate parties are unknown to the adversary.
(2) $\mathcal{S}$ generates a fresh server-random nonce for TLS, and [for TLS-1.3] a fresh Diffie-Hellman value.

To define context agreement, we first specify two properties: *successful completion* and *unique completion*. Successful completion states that for any initial strand that executes fully, there exists a complementary strand in the bundle that executes the protocol and agrees on a context. Unique completion states that within a bundle, no complementary strand exists that completes the protocol and does not agree on the context with the initial strand. We provide mathematical definitions for each of these properties below.

Within the following definitions, $\mathcal{B}$-height refers to the number of a strand's nodes that are in a bundle $\mathcal{B}$: a strand of integer height i has executed at least i steps of a protocol run. When executing the protocol between an initial and complementary strand, the complementary strand need not execute fully to satisfy the terms of the initial strand: the complementary strand's height can be less than a full run.

Definition 1 (Successful Completion)**.**
Let $\mathcal{B}$ be any bundle in a strand space $\mathcal{P}$, $\mathcal{I}$ and $\mathcal{R}$ be roles in $\mathcal{P}$, $\mathcal{X}$ be a session context for $\mathcal{P}$, and $\mathcal{O}$ be a set of origination assumptions for $\mathcal{I}$. Let i and j be any natural numbers.

$Succ(\mathcal{I}, \mathcal{R}, \mathcal{X}, \mathcal{O}, i, j)$ is true *if and only if (iff)* for all bundles $\mathcal{B} \in \mathcal{S}$ and strands $s \in \mathcal{I}$, there exists a strand $s' \in \mathcal{R}$ such that under $\mathcal{O}$, $s \in \mathcal{I}$ satisfies $\mathcal{X}$ with $\mathcal{B}$-height i and $s' \in \mathcal{R}$ satisfies $\mathcal{X}$ with $\mathcal{B}$-height j.

Definition 2 (Unique Completion)**.**
Let $\mathcal{I}$ and $\mathcal{R}$ be roles in a strand space $\mathcal{P}$, $\mathcal{X}$ be a session context in $\mathcal{P}$, and $\mathcal{O}$ be a set of origination assumptions for $\mathcal{I}$. Let i and j be any natural numbers.

$Uniq(\mathcal{I}, \mathcal{R}, \mathcal{X}, \mathcal{O}, i, j)$ is true iff $Succ(\mathcal{I}, \mathcal{R}, \mathcal{X}, \mathcal{O}, i, j) \wedge Succ(\mathcal{I}, \mathcal{R}', \mathcal{X}, \mathcal{O}, i, j) \implies \mathcal{R} = \mathcal{R}'$.

Using the properties and the assumption sets, we specify a context agreement security goal, which we call Goal 1. In Goal 1, a role holding a set of assumptions must demonstrate the successful completion and unique completion properties with a complementary role. Goal 1 is an authentication goal that requires *injective agreement* [Low97] between an initial and a complementary strand—there exists a one-to-one mapping of the session context between these strands. Any counterexample is sufficient to prove the failure of Goal 1. Counterexamples to Goal 1 describe attacks on the protocol under the initial role's perspective.

Goal 1 (Context Agreement)**.**
Let $\mathcal{P}$ be a strand space, $\mathcal{I}$ be an initial role in $\mathcal{P}$, $\mathcal{R}$ be a complementary role in $\mathcal{P}$, $\mathcal{O}$ be a set of origination assumptions for $\mathcal{I}$, and $\mathcal{X}$ be a session context for $\mathcal{P}$. Let $height(n) =$ the height of a fully executing strand in role n. Let i and j be natural numbers such that $i = height(\mathcal{I})$ and $j \leq height(\mathcal{R})$.

Context agreement is true iff $Succ(\mathcal{I}, \mathcal{R}, \mathcal{X}, \mathcal{O}, i, j) \wedge Uniq(\mathcal{I}, \mathcal{R}, \mathcal{X}, \mathcal{O}, i, j)$ is true.

In Section 8, we evaluate context agreement goals for each of our models from each role's perspective using CPSA.

8 CPSA Analysis of Security Goals

We state and prove theorems that evaluate context agreement goals for each of our models (Pre-SBP, SBP-TLS-1.2, SBP-mTLS-1.2, SBP-TLS-1.3, SBP-mTLS-1.3). For each

model, we prove a context agreement theorem for a distinct strand with a corresponding perspective. All strands (client, proxy, or server) use the corresponding perspectives and contexts that we detail in Section 7. While we do not mention intruder strands in our theorems, CPSA considers them when proving or disproving context agreement.

For our SBP models, context agreement implies prevention of cookie stealing and subsequent session hijacking by an adversary. Unique completion, one of the two requirements for context agreement, states that, for any legitimate client or proxy strand, no strand exists in a bundle that does not agree on the SBP context: the client's username, the proxy's identifier, and the correlation between the unbound and bound cookie. To make requests using the stolen cookie, the adversary must construct a bundle in which a client and a proxy strand exist, but hold non-matching contexts resulting from communicating with penetrator strands. When such a bundle exists, the adversary successfully manipulates a bound cookie from one SBP session to another to make requests, illustrating a successful cookie-stealing attack.

Table 1 summarizes the results of our analysis. For each strand represented by a column, we indicate for which of our models the strand satisfies a context agreement goal with its complementary strand (e.g., client and server, or client and proxy). In TLS-1.2 and TLS-1.3, the responder cannot authenticate the initiator within the DY model. In mTLS-1.2, the responder cannot guarantee freshness of the master secret; consequently, only SBP-mTLS-1.3 yields context agreement for all of its strands.

Each theorem asserts a context agreement goal as either true or false for a distinct strand and perspective. Proofs take one of two forms: an exhaustive proof that evaluates all possible, essentially different executions of the pre-SBP model, or a counterexample consisting of a CPSA shape that disproves the security goal. Termination of CPSA is only necessary for exhaustive proofs; a single counterexample is sufficient to prove context agreement false. For each model, we prove theorems for the model's two role perspectives. When a theorem proves context agreement false for a strand, we provide and discuss a counterexample that illustrates a protocol interaction by the adversary—in later sections, we discuss vulnerabilities, attacks, and risks that result from counterexamples.

Different TLS versions vary in their requirement for client certificates and guarantee of mutual freshness. Unlike mTLS-1.2 and mTLS-1.3, TLS-1.2 and TLS-1.3 do not require a client certificate because they provide only one-way authentication. In TLS-1.2 and mTLS-1.2, the initiator and responder contribute freshness via nonces, which are sent in the clear, exposing them to the adversary. By contrast, in TLS-1.3 and mTLS-1.3, the client and the server contribute additional freshness through a *Diffie-Hellman (DH)* key exchange, which does not leak that freshness. SBP-mTLS-1.3 requires client certificates and enables the proxy to guarantee mutual freshness of the master secret without leaking the freshness to the adversary; consequently, it is our only model to guarantee context agreement under both the proxy and the client origination assumptions.

Table 1: Summary of security goal analysis for the pre-SBP and SBP models. For each model, a check (✓) indicates that all strands holding the corresponding perspective satisfy context agreement (cookie stealing prevention) theorems. A crossmark (×) indicates that CPSA finds a counterexample that disproves context agreement for that perspective.

Model	client	proxy	server
Pre-SBP	×	N/A	×
SBP-TLS-1.2	✓	×	N/A
SBP-mTLS-1.2	✓	×	N/A
SBP-TLS-1.3	✓	×	N/A
SBP-mTLS-1.3	✓	✓	N/A

8.1 Method

To analyze SBP using CPSA, we: (1) model protocol roles by extracting messages and variables from a protocol specification, (2) specify origination assumptions for crucial variables that each role originates—often, these assumptions are specific to certain protocol perspectives and ultimately reside in skeletons, (3) specify skeletons for different role perspectives or special scenarios, (4) execute CPSA to produce output shapes, and (5) extract and prove security goals from the resulting shapes.

8.2 Pre-SBP Analysis

For the pre-SBP model, we specify and prove a theorem for strands holding the client and the server perspectives.

Theorem 1. *For an initial client role holding the client assumptions and a complementary server role in the Pre-SBP strand space, Goal 1 is false.*

Counterexample. For our Pre-SBP model, CPSA finds a single shape illustrating a counterexample in which an initial client strand fails to agree with a complementary server strand on the username associated with a cookie. This failure results from the adversary acquiring the cookie from a session with the server and reissuing it to the legitimate client in a separate session. Because the client can complete the protocol using a cookie issued for another user, this counterexample shows that stealing a cookie and using it to hijack a session is possible under the client's perspective in the Pre-SBP model. □

Theorem 2. *For an initial server role holding the server assumptions and a complementary client role in the Pre-SBP strand space, Goal 1 is false.*

Counterexample. CPSA finds a single counterexample shape in which the initial server strand completes the protocol entirely with penetrator strands owned by the adversary. In this shape, no legitimate client strand exists. Because the server has no assurance that the client's password is secret and has no other means by which to authenticate the client, the adversary completes arbitrary runs of the protocol with the server. As a result, there will never exist a legitimate complementary strand on which the server will agree to a Pre-SBP session context. □

8.3 SBP Analysis

As we did in the Pre-SBP model, we specify and prove theorems for the client and the proxy perspectives in the models of the original SBP and variations.

Theorem 3. *For an initial client role holding the client assumptions and a complementary proxy role in the SBP-TLS-1.2, SBP-mTLS-1.2, SBP-TLS-1.3, and SBP-mTLS-1.3 strand spaces, Goal 1 is true.*

Enumeration. For each SBP model, CPSA terminates and finds a single shape that satisfies context agreement between a client-a and a proxy-a strand. Because CPSA terminated and provably found all possible shapes, no counterexamples exist. □

Theorem 4. *For an initial proxy role holding the proxy assumptions and a complementary client role in the SBP-TLS-1.2, SBP-mTLS-1.2, and SBP-TLS-1.3 strand spaces, Goal 1 is false.*

Counterexample. For each of the three models, CPSA finds a single counterexample shape that illustrates the initial proxy strand completing the protocol with the adversary. Similar to the Pre-SBP model, this outcome results from the proxy's inability to authenticate the

client strands due to one of several reasons: In all three models, the client authenticates using a potentially stolen username and password pair. In SBP-TLS-1.2 and SBP-mTLS-1.2, the proxy has no assurance that the client generates a fresh pre-master secret—the client may provide a pre-master secret known to the adversary. In SBP-TLS-1.2 and SBP-TLS-1.3, the client does not produce a certificate for the proxy to authenticate the client. As a result, no bundle exists in which the proxy communicates with a legitimate client strand. □

Theorem 5. *For an initial proxy role holding the proxy assumptions and a complementary client strand in the SBP-mTLS-1.3 strand space, Goal 1 is true.*

Enumeration. CPSA terminates and finds a single shape in which a proxy-a strand completes the protocol and agrees on the SBP session context with a client-a strand. □

9 Vulnerabilities, Attacks, and Risks

9.1 Vulnerabilities

In decreasing order of severity, we list vulnerabilities exposed from our analysis.

The proxy cannot authenticate the client. In the SBP-TLS-1.2 and SBP-TLS-1.3 models, the proxy cannot authenticate the client; and the SBP-TLS-1.2 and SBP-mTLS-1.2 models lack mutual contribution to the freshness of the master secret. As a result, the proxy readily executes the SBP protocol with an adversary that presents stolen client credentials, requests using SBP cookies from compromised sessions, or requests originating from other SBP sessions.

A client may communicate with a corrupt proxy. Within the DY model, it is possible for a client to initiate the protocol with a corrupt proxy. In SBP, this scenario leaks the client's password and requests to the adversary, potentially enabling a man-in-the-middle attack against legitimate instances of SBP.

The server is an oblivious protocol participant. If an adversary communicates directly with a legacy server and bypasses the SBP proxy, they can pass stolen authentication cookies as we illustrate in the Pre-SBP model. This vulnerability is the result of the server being an *oblivious protocol participant*: the server participates in the SBP session without awareness of the SBP protocol. The legacy server's oblivious participation in SBP is an intentional design—a major goal of SBP is to avoid modifying the server.

The client cannot verify the binding of the cookie. Because the SBP proxy binds the cookie using a secret key unknown to the client, the client cannot verify the channel binding of the cookie. This property of SBP is by design: a major objective of SBP is to prevent the client from learning the unbound cookie and subsequently leaking it to an adversary. The primary issue resulting from this design is that the client may accept an inappropriately bound cookie, which may bind to a TLS channel in which the client does not participate, and use this cookie to make requests. It would be desirable for the client to be able to detect if they receive such a cookie and abort the protocol.

9.2 Attacks

We describe forwarding and tailgating attacks, which exploit the stated vulnerabilities.

Forwarding Attack. Because a client is unable to verify the binding of an SBP cookie, may communicate with a corrupt proxy, and cannot verify the binding a cookie, an adversary operating an SBP proxy can forward a client's credentials and requests in a concurrent session with a legitimate proxy. If a client reuses credentials for multiple servers, an adversary that receives the client's credentials on a corrupt proxy may establish an SBP session with a legitimate proxy by forwarding these credentials. Rather than forwarding a

client's credentials, the corrupt proxy can issue a cookie between itself and a legitimate proxy to the client, enabling the adversary to automatically forward the client's requests to the legitimate proxy. The potential consequences of forwarding are numerous: the adversary could authenticate using the client's credentials, present a client's legitimate requests as their own, and issue arbitrary responses to the client. In Appendix B, we illustrate an example of a forwarding attack from a proxy's perspective.

Tailgating Attack on SBP. Elsaad [AE22] showed that SBP is vulnerable to a "tailgating" attack, in which an adversary executes code inside of a client browser to forge requests on the client's SBP session rather than stealing the client's authentication cookie. Because these requests originate from the client on the appropriate TLS channel, and because they contain the corresponding bound authentication cookie, the proxy will accept the requests. While not possible within the DY model—a DY adversary cannot compromise a client in this manner—this attack is available within an adversarial model that the SBP authors specify: they allow one of their adversaries to execute arbitrary code in a client browser. Elsaad implemented a proof-of-concept of this attack against a custom SBP implementation [Els22].

9.3 Risks

We compare risks associated with legacy servers in typical environments, such as universities, with and without the protection of SBP.

Without SBP, the adversary poses a serious risk to a legacy server. By stealing authentication cookies from legitimate users and hijacking their sessions, the adversary gains access to accounts, which might range from basic user accounts to administrator accounts. Stealing cookies is simple against outdated servers and browsers and is possible for even unsophisticated adversaries to achieve. Upon gaining access to accounts, the adversary can impersonate the corresponding users, steal sensitive data to which the users have access, and perform actions using the user's privileges. If the account has elevated privileges, the adversary's access may result in a complete loss of the system—the adversary might freely take the system offline or exploit it to attack other systems on an organization's network.

When SBP wraps the legacy server, the adversary faces a significant obstacle: even when using known exploits to steal cookies, the adversary must find a way to breach a TLS connection or break the cookie's channel binding. Each of these tasks is challenging for all but the most sophisticated adversaries. Likely, it would be easier for the adversary instead to operate a malicious SBP proxy, harvest a user's credentials, and search for opportunities resulting from credential reuse. The risk of this attack depends on the properties of the TLS channel: an SBP proxy that authenticates users using mTLS-1.3 renders the user credentials significantly less valuable and requires an adversary to break the TLS channel. The damage of a successful attack is equivalent to that of breaching a legacy server that is unprotected by SBP.

10 SBP's Design Prevents Meaningful Improvement

SBP's requirement to mitigate cookie-based session hijacking without modifying the client or server software results in two serious limitations: the client cannot verify SBP's cookie channel bindings, and the SBP proxy learns the client password. SBP's core requirement (not to modify the client or server) represents a tradeoff: it makes SBP broadly applicable but limits its effectiveness at reducing security vulnerabilities.

Because clients believe they communicate with a legacy service and not an SBP proxy, they are unaware of the channel binding on their authentication cookie and thus cannot

authenticate the binding. This problem is impossible to correct without relaxing SBP's requirement not to modify client software.

The SBP proxy learns the client's password when forwarding it to the wrapped application, resulting in the user's password leaking to the adversary who compromises a legitimate proxy or operates a malicious proxy. There is no way to mitigate this issue without modifying the client and the proxy, such as by using *Password Authenticated Key Exchange (PAKE)* protocols [SLL+20].

11 Recommendations

We provide the following recommendations for SBP and other wrapper protocols.

Use Mutual TLS. To enable the server to authenticate the client, it would be helpful to use mTLS and require clients to use public-key certificates. Much of the Internet does not use mTLS, creating a large security liability, not only for SBP and other wrapper protocols, but for all protocols that rely on TLS for establishing authenticated end-to-end encryption. Furthermore, it is uncommon for individuals to possess certificates. From the server perspective, one-way TLS authentication often makes it impossible to achieve certain security goals due to the inability of the server to authenticate the client. Having clients use certificates and assuming them in protocol design can enable protocols to meet stronger security goals and can make it more difficult for adversaries to carry out adverse protocol interactions.

Prioritize Embedding. To limit exposure of the server, embed the proxy and server, and prevent any network communication other than the proxy's from reaching the server. Network administrators should be aware that an oblivious protocol participant will always enable protocol interactions within the DY model because it is possible for the oblivious protocol participant to communicate with the adversary. In situations where embedding is not possible, a dedicated VLAN (see Section 13) comprising only the proxy and the server can reduce exposure of the server.

Verify Bindings. The client should have some means by which to verify the binding of the authentication cookie. In SBP, it is a vulnerability that the client receives a bound authentication cookie for which it cannot verify the binding. Consequently, the client cannot determine to which SBP session, if any, the cookie is bound. Because the client should not learn the unbound cookie, the proxy could present the client with a zero-knowledge proof (see Section 13) of the binding's legitimacy for the SBP session.

12 Previous Work

To our knowledge, our work is the first formal-methods analysis of SBP.

In their original paper, Burgers et al. [BVvE13] informally evaluated SBP's security against a variety of known attacks against TLS and cookies. They consider attacks against six levels of a system executing SBP: (1) JavaScript, (2) HTML, (3) plug-ins, (4) browser, (5) TLS library, (6) operating system kernel. They conclude that SBP resists known cookie hijacking attacks against Levels 1 and 2, but fails to mitigate session hijacking when attacks target Level 3 (see the Tailgating Attack in Section 9.2) and higher. While technically correct, their analysis fails to consider the properties of the TLS channel necessary for a secure SBP session and does not state or prove any security properties.

Previous formal-methods analyses using CPSA include the CAVES multiparty attestation protocol [CGL+11], Zooko's forced-latency protocol [LZ17], the *Secure Remote Password (SRP)* protocol [SLL+20], TLS-1.3 [BZN21], the FIDO *Universal Authentication Framework (UAF)* [GSZ23, FHL22].

Like SBP, other designs for hardening flawed protocols running on legacy software frequently incorporate proxies. SessionShield [NMY+11] incorporate client-side proxies that prevent the client's browser from learning session identifiers, in turn making these unavailable to XSS attacks by an adversary. Pournaghshband et al. [PSR12] improve the security of mobile medical devices, which communicate wirelessly via flawed legacy protocols, by introducing a physical proxy that manages authenticated, encrypted sessions between these devices and their base stations. In the automotive space, works such as Wang and Sawhney's VeCure [WS14] and Schmandt et al.'s Mini-MAC [SSB17] augment the existing protocols of legacy controller area networks (CAN buses) to mitigate serious flaws. As with SBP, formal-methods analyses of these systems were not part of their design process and often do not exist.

Other approaches for hardening legacy software include migrating these systems to the cloud [ARM+15], encapsulation [Sne00], remote attestation via trusted platform modules [SWP08], and virtualization [PNS07].

13 Discussion

We discuss several issues raised by our work, including the benefits of isolation in wrapper solutions, inadequate adversarial models, authenticating via passwords, SBP's custom channel binding, requirements for effective cryptographic binding, and open problems.

Benefits of Isolation. Within the DY model, legacy systems that participate in protocols obliviously are vulnerable to adversaries that bypass network controls to communicate with the systems directly. To mitigate this vulnerability, one can isolate the systems physically and logically. To isolate the system physically, configure the network so that it physically communicates only with a proxy executing the wrapping protocol. To isolate the system logically, establish a VLAN [LT07] to separate network traffic logically. The objective is to prevent any network entity other than an authorized proxy from communicating with the wrapped legacy system.

Inadequate Adversarial Models. When designing and analyzing protocols, researchers should use well-defined adversarial models and formal-methods tools. Without a well-defined adversarial model, it is impossible to articulate and analyze the security properties. The SBP authors [BVvE13] did not analyze their protocol using any formal model such as the DY model. Although their analysis identifies several attacks and mitigations, they did not discuss limitations within the DY model, such as the client communicating with a corrupt proxy, the server being unable to authenticate the client, and the client's inability to verify the binding of a cookie (see Section 9). Furthermore, experience has shown that, without the aid of formal-methods tools, protocol design and analysis is highly error-prone.

Authenticating via Passwords. SBP authenticates users using traditional passwords, which are often a poor choice for authentication. Users often rely on weak passwords, store passwords insecurely, and reuse passwords across multiple services. Passwords also suffer from poor storage practices (e.g., storing plain text passwords in a database) and precomputation by adversaries. In the strand space model, an adversary may acquire a user's password in several ways: a legitimate strands may transfer the password to penetrator strands that masquerade as a legitimate service or the adversary compromises a password that a user registers with multiple services.

SBP's Channel Binding vs. Channel Binding Standards. Rather than relying on existing TLS channel binding standards, which bind to communication endpoints to facilitate cross-session bindings, SBP introduces a custom TLS channel binding in which messages bind directly to a TLS session's master secret. A consequence of this binding is that bound cookies in SBP become invalid when communicants negotiate a new TLS channel, but are also impossible to transfer between different TLS sessions. The following limitation would result if SBP bound cookies to endpoints: if multiple users shared one

end-point, it might be possible for an adversary to transfer cookies for different users between sessions that share common endpoints. Building wrapper protocols such as SBP using standard TLS channel bindings requires additional study and consideration.

Requirements for Cryptographic Binding. Effective cryptographic bindings must achieve several goals: (1) Each role in a protocol is able to verify, append, and sign a session context. (2) The bindings are unique to a protocol instance, version, and session; and (3) upon completing execution, each legitimate protocol role must agree on the context. In practice, many protocols, including SBP, fail to bind in a manner that achieves these goals. Oblivious protocol participants are unable to participate in effective binding because they are, by design, unaware of the protocol context. Other protocols fail to bind values consistently, neglecting to bind crucial values such as challenges or nonces, or binding in a manner that prevents other legitimate protocol roles from verifying the bindings.

Using zero-knowledge proofs [GMR19], protocol roles can potentially verify bindings without disclosing sensitive values, such as the unbound cookie in SBP.

Open Problems and Future Work. The benefits and drawbacks of enhancing systems with wrapper solutions demands further study. Our analysis of SBP and oblivious protocol participants underscores the needs to specify best practices for designing and implementing wrapper protocols, and to devise better bindings for such protocols. Our work raises awareness of cryptographic binding and formal-methods verification for practitioners working to wrap flawed systems. As future work, we are developing tools for applying cryptographic bindings to protocol specifications automatically.

14 Conclusion

We performed a formal-methods analysis of four models of SBP, the original SBP protocol and three variations, and compared them with a baseline model without SBP. In the original SBP model and variations other than mTLS-1.3, the proxy's cryptographic perspective does not guarantee resistance to cookie stealing by an adversary because of several issues. Except for mTLS-1.3 connections, the proxy cannot authenticate the client, or the adversary learns the freshness nonces. Passwords are insufficient to authenticate users in the DY model. These issues result in part from a lack of adoption of client certificates, design flaws in TLS-1.2 and mTLS-1.2, and SBP's requirement to conform to legacy password authentication mechanisms. To our knowledge, we are the first to analyze SBP using formal methods and the first to highlight the phenomenon of oblivious protocol participants in wrapper protocols.

To address each of these issues, we recommend binding to a mTLS-1.3 channel. To accommodate this recommendation, the Internet must move towards mutual authentication between all communicants and rely on alternate authentication mechanisms, such as multifactor authentication or user-bound certificates, rather than passwords. Despite limitations of the original SBP, we find that the protocol mitigates session hijacking from the client's perspective and, in many applications, significantly complicates an adversary's attempts to hijack sessions via cookie stealing.

With many organizations struggling to protect flawed legacy systems on their networks, wrapping protocols are inevitable and necessary. In the DY model, such protocols must take great care to establish and bind to encrypted channels with the necessary properties. Where possible, these protocols should eliminate flawed authentication mechanisms, such as traditional passwords, in favor of more robust solutions. Because it is often not possible or practical to modify the legacy systems or the client software that interacts with them, wrapping protocols such as SBP may represent the most attractive practical mitigation to known flaws. Our formal-methods analysis sheds insights on the strengths and limitations of wrapping protocols that depend on channel binding.

References

[AE22] Kirellos N. Abou Elsaad. *A Formal Methods Analysis of the Session Binding Proxy Protocol*. PhD thesis, University of Maryland, Baltimore County, 2022. URL: http://proxy-bc.researchport.umd.edu/login?url=https://www.proquest.com/dissertations-theses/formal-methods-analysis-session-binding-proxy/docview/2720939643/se-2.

[AN96] Martín Abadi and Roger M. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Trans. Software Eng.*, 22(1):6–15, 1996. doi:10.1109/32.481513.

[ARM+15] Luis Márquez Alcañiz, David Garcia Rosado, Haralambos Mouratidis, Daniel Mellado, and Eduardo Fernández-Medina. A framework for secure migration processes of legacy systems to the cloud. In Anne Persson and Janis Stirna, editors, *Advanced Information Systems Engineering Workshops - CAiSE 2015 International Workshops, Stockholm, Sweden, June 8-9, 2015, Proceedings*, volume 215 of *Lecture Notes in Business Information Processing*, pages 507–517. Springer, 2015. doi:10.1007/978-3-319-19243-7_46.

[Bla13] Bruno Blanchet. Automatic verification of security protocols in the symbolic model: The verifier ProVerif. In Alessandro Aldini, Javier López, and Fabio Martinelli, editors, *Foundations of Security Analysis and Design VII - FOSAD 2012/2013 Tutorial Lectures*, volume 8604 of *Lecture Notes in Computer Science*, pages 54–87. Springer, 2013. doi:10.1007/978-3-319-10082-1_3.

[BVvE13] Willem Burgers, Roel Verdult, and Marko C. J. D. van Eekelen. Prevent session hijacking by binding the session to the cryptographic network credentials. In Hanne Riis Nielson and Dieter Gollmann, editors, *Secure IT Systems - 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings*, volume 8208 of *Lecture Notes in Computer Science*, pages 33–50. Springer, 2013. doi:10.1007/978-3-642-41488-6_3.

[BZN21] Prajna Bhandary, Edward Zieglar, and Charles Nicholas. Searching for selfie in TLS 1.3 with the Cryptographic Protocol Shapes Analyzer. In Daniel Dougherty, José Meseguer, Sebastian Alexander Mödersheim, and Paul D. Rowe, editors, *Protocols, Strands, and Logic - Essays Dedicated to Joshua Guttman on the Occasion of his 66.66th Birthday*, volume 13066 of *Lecture Notes in Computer Science*, pages 50–76. Springer, 2021. doi:10.1007/978-3-030-91631-2_3.

[CGL+11] George Coker, Joshua D. Guttman, Peter A. Loscocco, Amy L. Herzog, Jonathan K. Millen, Brian O'Hanlon, John D. Ramsdell, Ariel Segall, Justin Sheehy, and Brian T. Sniffen. Principles of remote attestation. *Int. J. Inf. Sec.*, 10(2):63–81, 2011. URL: https://doi.org/10.1007/s10207-011-0124-7, doi:10.1007/S10207-011-0124-7.

[DY83] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983. doi:10.1109/TIT.1983.1056650.

[Els22] Kirellos Abou Elsaad. SBP Tailgating Attack Implementation. https://github.com/kmanayer/sbp_impl, 2022. Accessed: 2023-08-14.

[EMM07] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design V, FOSAD 2007/2008/2009 Tutorial Lectures*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2007. doi:10.1007/978-3-642-03829-7_1.

[FHL22] Jonathan Fuchs, Sophia Hamer, and Danning Liu. A man-in-the-middle attack on the FIDO UAF registration protocol. CSEE Dept, UMBC, CMSC 491/691 report, December 2022.

[GLRR] J.D. Guttman, M.D. Liskov, J.D. Ramsdell, and P.D. Rowe. The Cryptographic Protocol Shapes Analyzer (CPSA). https://github.com/mitre/cpsa.

[GMR19] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. In Oded Goldreich, editor, *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 203–225. ACM, 2019. doi:10.1145/3335741.3335750.

[GSZ23] Enis Golaszewski, Alan T. Sherman, and Edward Zieglar. Cryptographic binding should not be optional: A formal-methods analysis of FIDO UAF authentication. unpublished manuscript, CSEE Dept, UMBC, July 2023.

[Gut00] Joshua D. Guttman. Security Goals: Packet trajectories and strand spaces. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design, Tutorial Lectures [revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000]*, volume 2171 of *Lecture Notes in Computer Science*, pages 197–261. Springer, 2000. doi:10.1007/3-540-45608-2_4.

[Gut14] Joshua D. Guttman. Establishing and preserving protocol security goals. *J. Comput. Secur.*, 22(2):203–267, 2014. doi:10.3233/JCS-140499.

[JDG+15] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 579–593. USENIX Association, 2015. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/jagpal.

[KFS+22] Ranjita Pai Kasturi, Jonathan Fuller, Yiting Sun, Omar Chabklo, Andres Rodriguez, Jeman Park, and Brendan Saltaformaggio. Mistrust plugins you must: A large-scale study of malicious plugins in WordPress marketplaces. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 161–178. USENIX Association, 2022. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/kasturi.

[KGC+14] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 641–654, San Diego, CA, August 2014. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/kapravelos.

[KSW97] John Kelsey, Bruce Schneier, and David A. Wagner. Protocol interactions and the chosen protocol attack. In Bruce Christianson, Bruno Crispo, T. Mark A. Lomas, and Michael Roe, editors, *Security Protocols, 5th International Workshop, Paris, France, April 7-9, 1997, Proceedings*, volume 1361 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 1997. URL: https://doi.org/10.1007/BFb0028162, doi:10.1007/BFB0028162.

[Low95] Gavin Lowe. An attack on the Needham-Schroeder Public-Key Authentication Protocol. *Inf. Process. Lett.*, 56(3):131–133, 1995. doi:10.1016/0020-0190(95)00144-2.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *10th Computer Security Foundations Workshop Proceedings*, pages 31–43. IEEE CS Press, 1997.

[LRGR16] Moses D Liskov, John D Ramsdell, Joshua D Guttman, and Paul D Rowe. *The Cryptographic Protocol Shapes Analyzer: A Manual*. The MITRE Corporation, 2016.

[LRT11] Moses Liskov, Paul Rowe, and Javier Thayer. Completeness of CPSA. Technical report, MITRE, 2011. https://www.mitre.org/sites/default/files/pdf/12_0038.pdf.

[LT07] Garrett Leischner and Cody Tews. Security through VLAN segmentation: Isolating and securing critical assets without loss of usability. In *Proceedings of the 9th Annual Western Power Delivery and Automation Conference, Spokane, WA*, 2007.

[Luo98] Ari Luotonen. *Web proxy servers*. Prentice-Hall, Inc., 1998.

[LZ17] EF Lanus and EV Zieglar. Analysis of a forced-latency defense against man-in-the-middle attacks. *Journal of Information Warfare*, 16(2):66–78, 2017.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013. doi:10.1007/978-3-642-39799-8_48.

[NMY+11] Nick Nikiforakis, Wannes Meert, Yves Younan, Martin Johns, and Wouter Joosen. SessionShield: Lightweight protection against session hijacking. In Úlfar Erlingsson, Roel J. Wieringa, and Nicola Zannone, editors, *Engineering Secure Software and Systems - Third International Symposium, ESSoS 2011, Madrid, Spain, February 9-10, 2011. Proceedings*, volume 6542 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2011. doi:10.1007/978-3-642-19125-1_7.

[NS78] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, December 1978. URL: http://doi.acm.org/10.1145/359657.359659, doi:10.1145/359657.359659.

[Pau98] Lawrence C. Paulson. Inductive analysis of the internet protocol TLS. In Bruce Christianson, Bruno Crispo, William S. Harbison, and Michael Roe, editors, *Security Protocols, 6th International Workshop, Cambridge, UK,*

April 15-17, 1998, Proceedings, volume 1550 of *Lecture Notes in Computer Science*, pages 13–23. Springer, 1998. doi:10.1007/3-540-49135-X_2.

[PNS07] Shaya Potter, Jason Nieh, and Matt Selsky. Secure isolation of untrusted legacy applications. In *LISA*, volume 7, pages 1–14, 2007.

[PSR12] Vahab Pournaghshband, Majid Sarrafzadeh, and Peter L. Reiher. Securing legacy mobile medical devices. In Balwant Godara and Konstantina S. Nikita, editors, *Wireless Mobile Communication and Healthcare - Third International Conference, MobiHealth 2012, Paris, France, November 21-23, 2012, Revised Selected Papers*, volume 61 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 163–172. Springer, 2012. doi:10.1007/978-3-642-37893-5_19.

[PT17] Min-Seok Pang and Huseyin Tanriverdi. Security breaches in the US federal government. In *Workshop on the Economics of Information Security*, 2017.

[Ram15] John D Ramsdell. CPSA and formal security goals. *The MITRE Corporation*, 2015.

[Res01] Eric Rescorla. *SSL and TLS: Designing and building secure systems*. Addison-Wesley, 2001.

[RGL16] Paul D. Rowe, Joshua D. Guttman, and Moses D. Liskov. Measuring protocol strength with security goals. *Int. J. Inf. Sec.*, 15(6):575–596, 2016. URL: https://doi.org/10.1007/s10207-016-0319-z, doi:10.1007/S10207-016-0319-Z.

[RTFB20] Germán E. Rodríguez, Jenny G. Torres, Pamela Flores, and Diego E. Benavides. Cross-Site scripting (XSS) attacks and mitigation: A survey. *Computer Networks*, 166:106960, 2020. URL: https://www.sciencedirect.com/science/article/pii/S1389128619311247, doi:https://doi.org/10.1016/j.comnet.2019.106960.

[SLL+20] Alan T. Sherman, Erin Lanus, Moses Liskov, Edward Zieglar, Richard Chang, Enis Golaszewski, Ryan Wnuk-Fink, Cyrus J. Bonyadi, Mario Yaksetig, and Ian Blumenfeld. Formal methods analysis of the Secure Remote Password protocol. In Vivek Nigam, Tajana Ban Kirigin, Carolyn L. Talcott, Joshua D. Guttman, Stepan L. Kuznetsov, Boon Thau Loo, and Mitsuhiro Okada, editors, *Logic, Language, and Security - Essays Dedicated to Andre Scedrov on the Occasion of His 65th Birthday*, volume 12300 of *Lecture Notes in Computer Science*, pages 103–126. Springer, 2020. doi:10.1007/978-3-030-62077-6_9.

[SN20] Hamza Saleem and Muhammad Naveed. SoK: Anatomy of data breaches. *Proc. Priv. Enhancing Technol.*, 2020(4):153–174, 2020. URL: https://doi.org/10.2478/popets-2020-0067, doi:10.2478/POPETS-2020-0067.

[Sne00] Harry M. Sneed. Encapsulation of legacy software: A technique for reusing legacy software components. *Ann. Softw. Eng.*, 9:293–313, 2000. doi:10.1023/A:1018989111417.

[SSB17] Jackson Schmandt, Alan T Sherman, and Nilanjan Banerjee. Mini-MAC: Raising the bar for vehicular security with a lightweight message authentication protocol. *Vehicular Communications*, 9:188–196, 2017.

[SWP08] Dries Schellekens, Brecht Wyseur, and Bart Preneel. Remote attestation on legacy operating systems with trusted platform modules. *Sci. Comput. Program.*, 74(1-2):13–22, 2008. URL: https://doi.org/10.1016/j.scico.2008.09.005, doi:10.1016/J.SCICO.2008.09.005.

[THG99] F. Javier Thayer, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *J. Comput. Secur.*, 7(1):191–230, 1999. URL: https://doi.org/10.3233/jcs-1999-72-304, doi:10.3233/JCS-1999-72-304.

[UMB23] UMBC Protocol Analysis Lab. PAL GitHub repository. https://tinyurl.com/3d2wnhuf, October 2023.

[vEMHV13] MCJD van Eekelen, R Ben Moussa, Engelbert Hubbers, and Roel Verdult. Blackboard security assessment. Technical report, Nijmegen: Institute for Computing and Information Sciences, 2013.

[WKDP14] Nicholas Weaver, Christian Kreibich, Martin Dam, and Vern Paxson. Here be web proxies. In Michalis Faloutsos and Aleksandar Kuzmanovic, editors, *Passive and Active Measurement - 15th International Conference, PAM 2014, Los Angeles, CA, USA, March 10-11, 2014, Proceedings*, volume 8362 of *Lecture Notes in Computer Science*, pages 183–192. Springer, 2014. doi:10.1007/978-3-319-04918-2_18.

[WS14] Qiyan Wang and Sanjay Sawhney. Vecure: A practical security framework to protect the CAN bus of vehicles. In *4th International Conference on the Internet of Things, IOT 2014, Cambridge, MA, USA, October 6-8, 2014*, pages 13–18. IEEE, 2014. doi:10.1109/IOT.2014.7030108.

[ZE10] Yuchen Zhou and David Evans. Why aren't HTTP-only cookies more widely deployed? *Proceedings of 4th Web*, 2, 2010.

A Acronyms and Abbreviations

CPSA	Cryptographic Protocol Shapes Analyzer
DY	Dolev-Yao
FIDO	Fast Identity Online
HMAC	Hashed-Based Message Authentication Code
HTTP	Hypertext Transfer Protocol
NS	Needham-Schroeder
SBP	Session Binding Proxy Protocol
TLS	Transport Layer Security
UAF	Universal Authentication Framework
VLAN	Virtual Local Area Network
XSS	Cross-Site Scripting

B Forwarding Attack

Figure 4 illustrates the forwarding attack described in Section 9.2.

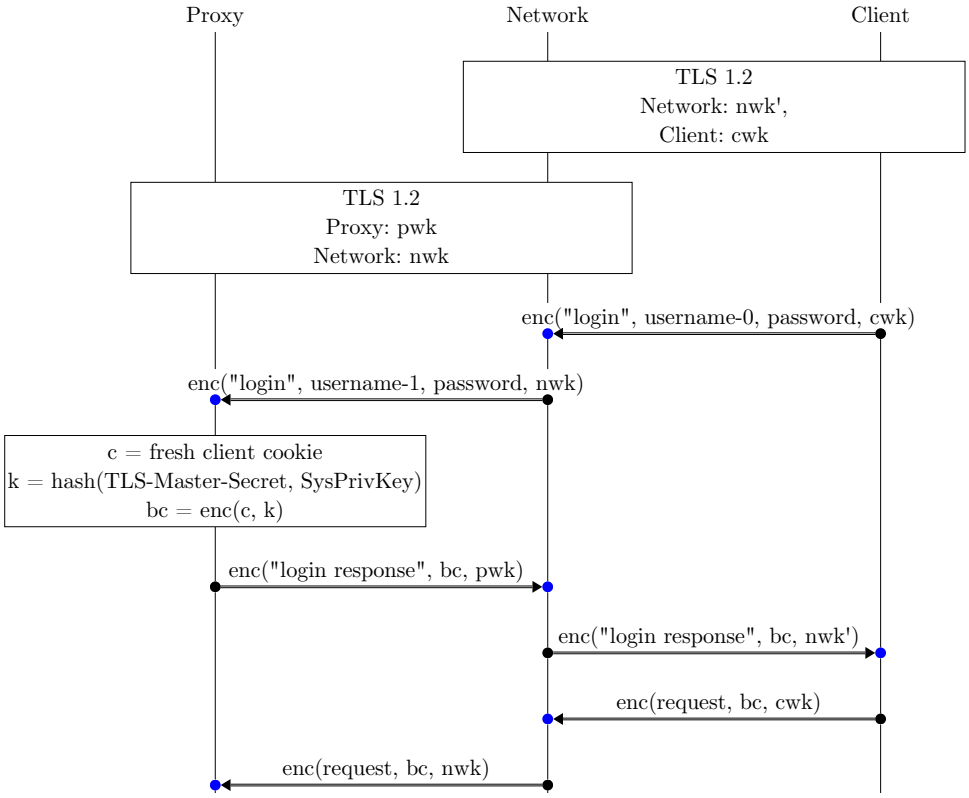

Figure 4: Protocol diagram illustrating two types of forwarding in a forwarding attack: credential forwarding and request forwarding. First, the network exploits the client's reuse of a password to authenticate with a legitimate proxy while posing as an SBP proxy to the client. Second, independently of the first exploit, the network reuses the proxy's bound cookie in its session with the client, enabling it to forward the client's requests automatically without any modification beyond decryption and encrypting with the correct key. These issues result from the proxy's inability to authenticate the client, the client's inability to verify the binding of the cookie, and the client's potential to communicate with a corrupt proxy in the DY model.