ITC 3/50

Information Technology and Control
Vol. 50 / No. 3 / 2021
pp. 570-587
DOI 10.5755/j01.itc.50.3.27844

Automated State-Machine-Based Analysis of Hostname Verification in IPsec Implementations

Received 2020/10/12 | Accepted after revision 2021/04/15

http://dx.doi.org/10.5755/j01.itc.50.3.27844

HOW TO CITE: Guo, J., Gu, C., Chen, X., Lu, S., Wei, F. (2021). Automated State-Machine-Based Analysis of Hostname Verification in IPsec Implementations. *Information Technology and Control*, 50(3), 570-587. https://doi.org/10.5755/j01.itc.50.3.27844

Automated State-Machine-Based Analysis of Hostname Verification in IPsec Implementations

Jiaxing Guo, Chunxiang Gu, Xi Chen, Siqi Lu, Fushan Wei

Henan Key Laboratory of Network Cryptography Technology, Zhengzhou, China;
e-mails: guojiaxing124lab@gmail.com; Chunxianggu.meac@gmail.com; xycuckoo@tsinghua.org.cn; 080lusiqi@sina.com; weifs831020@163.com

Corresponding author: Chunxianggu.meac@gmail.com

Owing to the advent and rapid development of Internet communication technology, network security protocols with cryptography as their core have gradually become an important means of ensuring secure communications. Among numerous security protocols, certificate authentication is a common method of identity authentication, and hostname verification is a critical but easily neglected process in certificate authentication. Hostname verification validates the identity of a remote target by checking whether the hostname of the communication partner matches any name in the X.509 certificate. Notably, errors in hostname verification may cause security problems with regard to identity authentication. In this study, we use a model-learning method to conduct security testing for hostname verification in internet protocol security (IPsec). This method can analyze the problems entailed in implementing hostname verification in IPsec by effectively inferring the deterministic finite automaton model that can describe the matching situation between the certificate subject name and the hostname for different rules. We analyze two popular IPsec implementations, Strongswan and Libreswan, and find five violations. We use some of these violations to conduct actual attack tests on the IPsec implementation. The results show that under certain conditions, attackers can use these flaws to carry out identity impersonation attacks and man-in-the-middle attacks.

KEYWORDS: IPsec hostname verification, state machine inference, protocol security analysis, model learning, identity impersonation attack.

1. Introduction

Owing to the rapid development of network communication technology and the continuous upgrading of the Internet industry, Internet products and services have gradually become an important part of people's daily lives. However, with the gradual improvements in relevant technologies, network security problems, such as data leakage, high-risk vulnerabilities, and network attacks, are also threatening individual privacy, social stability, and even national security. As society's dependence on networks continues to increase, ensuring communication security has become an important problem that must be solved urgently.

In network security communication, a common method to verify user identity is through digital certificates entailing the application of the public-key cryptographic algorithm. A trusted third party issues a certificate. One party in the communication uses its own private key to sign the message, and the other party uses the corresponding public key to verify the message. The subject name corresponding to the certificate is compared with the user identity to ensure that the identity of the certificate provider is legal.

As one of the most important security protocols, internet protocol security (IPsec) is widely used in virtual private network (VPN) and IPv6 services, aiming to provide users with secure end-to-end communications. In view of the risks that the internet protocol (IP) layer may face, IPsec provides the three security services of identity authentication, confidentiality, and integrity (including data-source authentication and integrity verification) and resists replay attacks. The communication process of IPsec is divided into two main stages: negotiation and security data transmission. In the negotiation phase, the two communicating parties perform the negotiation of cryptographic parameters, identity authentication, and session-key generation. In the subsequent secure data transmission, the negotiated cryptographic algorithm and session key are used to protect the interactive data. Among the multiple identity authentication methods defined by IPsec, certificate authentication is a recommended method. In the certificate authentication mode, the communication party not only uses the public key provided by the other party to verify its signed message, but also matches the hostname of the other party with the subject name in the public-key certificate to validate the ownership of the public-key certificate and the identity of the other party. In addition, the search and selection of security policy database (SPD) policies rely on hostname verification in IPsec. Therefore, the inaccuracy of hostname verification may hinder the security of IPsec authentication.

Because of numerous special circumstances, hostname verification is a complicated process. For example, in hostname matching, multiple identity types (IP, distinguished name (DN), fully qualified domain name (FQDN), email, etc.) and various special aspects (common name/alternate name matching order, wildcards, null bytes, etc.) must be considered. Although hostname verification is critical to protocol security, a majority of the work related to certificate verification entails adversarial tests on the secure socket layer/transport layer security (SSL/TLS) certificates, and there are only a few studies involving IPsec hostname verification tests. Hostname verification is essentially a string-matching problem and is very similar to regular-expression matching under complex conditions. For the subject name of a given certificate, all the hostnames that match it can be regarded as a specific general language family or matching set. Therefore, in this study, we use a model-learning method to infer the deterministic finite automata (DFA) model, which can describe the matching situation between the certificate subject name and the hostname in different rules effectively; further, it can find the implementation problem of IPsec hostname verification by analyzing the DFA model.

Using this method, we analyze the two popular IPsec implementations, Strongswan and Libreswan, and find five violations: 1) incorrect handling of space characters in ID_DER_ASN1_DN type IDs, 2) incorrect handling of case-insensitive strings in ID_DER_ASN1_DN type IDs, 3) incorrect matching of wildcards and substrings in ID_FQDN type IDs, 4) incorrect matching of substrings in ID_USER_FQDN type IDs, and 5) unsupported null-byte certificate matching. We use the violations, 2) and 3), to conduct actual attack tests on the IPsec implementation. The results show that when the legal private and certificate chain are configured, attackers can use these flaws to carry out identity impersonation attacks and man-in-the-middle attacks.

The overall contributions of this study are summarized as follows:

1 We designed and implemented the automated state machine inference of the implementation of hostname verification for IPsec. To the best of our knowledge, there have only been a few detailed and relevant analyses on IPsec hostname verification before;
2 We employed this method to analyze two IPsec hostname verification implementations and found five violations;
3 We used some of the detected flaws to implement identity impersonation attacks and man-in-the-middle attacks, thereby demonstrating that our work contributes to finding security vulnerabilities.

The remainder of this article is organized as follows: related work is introduced in the Section 2; background and basic knowledge are explained in the Section 3. In Section 4, we discuss the principle and framework of the model-learning method. In Section 5, we present the analysis results obtained using this method and the verification of the actual attack. Finally, we provide the conclusions of this study in Section 6.

2. Related Work

In 1998, Harkins and Carrell [12] proposed the IKEv1 protocol based on ISAKMP, Oakley, and SKEME. Since then, IPsec based on IKEv1, authentication header (AH), and encapsulating packet (ESP) has attracted increasing attention from researchers. In 2005, Kaufman [20] proposed IKEv2, which caused a greater wave of research on internet key exchange (IKE) protocol. In 2006, Eronen and Hoffman [8] described the establishment of IKEv2 interoperable implementations (request for comments RFC 4718). In September 2010, Kaufman et al. [18] proposed an updated version of IKEv2 (RFC 5996) based on RFC 4718, and it contained a detailed description of RFC 4718. In October 2014, the updated IKEv2 (RFC 7296) was standardized to achieve better security and accuracy [19].

In view of the importance of IPsec, in recent years, researchers have conducted a series of security analyses on IPsec, including formal analysis, security proof, model checking, fuzzing, symbolic execution, and state machine inference. In terms of protocol documents, Cremers [6] conducted a large-scale formal analysis of IKEv1 and IKEv2 protocols and reported that IKEv1 and IKEv2 cannot satisfy strong authentication and are vulnerable to reflection attacks. Patel and Jinwala [29] used the colored Petri net model to conduct automatic formal analysis of denial of service (DoS) attacks on the IKEv2 protocol and stated that the IKEv2 protocol is vulnerable to DoS attacks owing to the use of expensive encryption operations to derive the shared key and transmit it securely. Cheng et al. [4] conducted a formal analysis of the IKEv3 draft and found that the IKEv3 protocol is vulnerable to reflection attacks and DoS attacks. Nussbaumer [28] used EasyCrypt to evaluate the security of the IKEv2 protocol and showed that the IKEv2 protocol has semantic security under the authentication key exchange model. In terms of specific protocol implementations, Ninet et al. [26] used Spin to perform model checking on the IKEv2 protocol and pointed out that the reflection attack found in [6] was not applicable in practice; however, in another study, Ninet et al. [27] designed a deviation attack against IKEv2 that used the penultimate authentication flaw found in [6] to perform a DoS attack. Yang et al. [35] conducted a fuzzing test against the IKEv1 protocol and designed a tool, IKEProFuzzer. They constructed test cases by modifying specific fields and designed corresponding exception detectors for different applications. Finally, five new vulnerabilities were discovered in six routers and applications. Using a similar method, Cui et al. [7] conducted a fuzz test on an implementation of Cisco ASA in the IKEv2 protocol and discovered security problems such as buffer overflow and DoS. Felsch et al. [10] proposed the principle of Bleichenbacher oracle attacks against IPsec, showing that it was sufficient to break all the certificate authentication schemes of the IKEv1 and IKEv2 protocols, and discovered specific Bleichenbacher oracle attack vulnerabilities in four router firmware. Chau et al. [3] used symbolic execution technology to analyze 15 open-source implementations (including IPsec) of the PKCS#1 v1.5 signature verification code and found six semantic errors. They also identified that four corresponding implementations were vulnerable to new variants of Bleichenbacher's low-index RSA signature forgery. In a previous work [11], we combined model-learning and model-checking methods to analyze the execution logic of three IKEv2 implementations. By analyzing the DFA model, we found three violations of the RFCs.

However, none of the aforementioned studies focused on the correctness of IPsec hostname matching. In contrast, a few studies have been conducted on hostname matching in SSL/TLS protocol. Kaminsky et al. [17] summarized recent attacks on the SSL/TLS certificate authentication architecture, showing that in some hostname verification implementations, error handling of null characters embedded in X.509 certificates can induce a certificate authority (CA) to issue valid leaf certificates with incorrect subject names. Fahl et al. [9] developed a tool for static analysis of Android code to detect whether the target implementation has problems such as accepting self-signed certificates or not verifying hostnames. They found that a large part of the application contains related flaws, which are vulnerable to man-in-the-middle attacks. Sounthiraraj et al. [33] improved the analysis technology proposed in [9] and developed a tool called SMV hunter. The tool optimizes the static analysis source code function and adds dynamic testing, thereby improving the detection of defects in error verification in basic TLS certificate. Sivakorn et al. [31] focused on the hostname verification process and developed a test tool named HVLearn based on the state-machine learning algorithm. They inferred the corresponding state machine model to find differences in the implementation of specific hostname verification. They found eight unique violations of RFC specifications in eight TLS implementations, some of which may allow man-in-the-middle attacks. Inspired by these works, in this study, we analyze the hostname verification module of IPsec.

3. Preliminaries

In this section, we introduce the basic framework and negotiation process of IPsec, fundamental structure of the X.509 digital certificate, types of IPsec identification that can be used, and relevant policy of IPsec hostname matching.

3.1. IPsec Negotiation Process

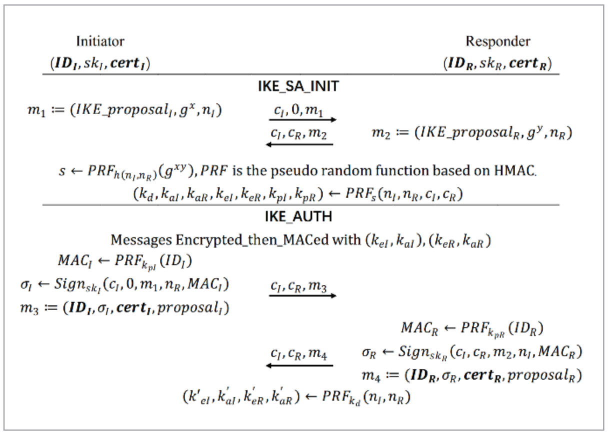
IPsec is a security protocol that runs on the IP layer, and it can be used for end-to-end confidential communication. To mitigate the security risks that the IP layer may face, IPsec provides three security services: confidentiality, integrity verification, and identity authentication, and resists replay attacks. The IPsec process is divided into negotiation and communication phases. In the negotiation phase (IKE protocol), the communication parties negotiate cryptographic parameters, identity authentication, and session key generation. In the subsequent secure communication (ESP or AH protocol), the negotiated cryptographic algorithm and session key are applied to protect the interactive data. Currently, the IKE protocol has two versions, IKEv1 (1998) and IKEv2 (2005). Although the IKEv2 protocol is designed to officially phase out the IKEv1 version, most operating systems and network devices still support the configuration and use of both versions. The IKEv1 protocol contains four authentication methods: signature-based authentication, public-key encryption, revised public-key encryption, and pre-shared-key-based authentication; the first three require digital certificate participation. The IKEv2 protocol omits two authentication methods based on public-key encryption and only retains the signature-based and pre-shared-key authentication.

Compared with pre-shared keys that are easy to leak or suffer from dictionary attacks, certificate authentication has higher security; therefore, it is recommended for use in various operating systems and communication devices. In the certificate authentication mode, one party in the communication uses its own private key to sign a specific message, and the other party uses the corresponding public key to verify the message. Both parties rely on the dependence of the public–private key pair to ensure authentication. In addition, in IPsec, both parties in communication need to verify the match between the hostname of the other party and the subject name in the public-key certificate to ensure the ownership of the public key and the legitimacy of the identity of the other party.

Figure 1 shows the communication flow of the digital signature authentication in IKEv2 protocol. The two parties in the communication first complete the negotiation of the IKE proposal (protocol version,

Figure 1

IKEv2 process of signature-based authentication


Initiator
$(\boldsymbol{ID_I}, sk_I, \boldsymbol{cert_I})$
Responder
$(\boldsymbol{ID_R}, sk_R, \boldsymbol{cert_R})$
IKE_SA_INIT
$m_1 := (IKE\_proposal_I, g^x, n_I)$
$c_I, 0, m_1$
c_I, c_R, m_2
$m_2 := (IKE\_proposal_R, g^y, n_R)$
$s \leftarrow PRF_{h(n_I, n_R)}(g^{xy})$, PRF is the pseudo random function based on HMAC.
$(k_d, k_{aI}, k_{aR}, k_{eI}, k_{eR}, k_{pI}, k_{pR}) \leftarrow PRF_s(n_I, n_R, c_I, c_R)$
IKE_AUTH
Messages Encrypted_then_MACed with $(k_{eI}, k_{aI}), (k_{eR}, k_{aR})$
$MAC_I \leftarrow PRF_{k_{pI}}(ID_I)$
$\sigma_I \leftarrow Sign_{sk_I}(c_I, 0, m_1, n_R, MAC_I)$
c_I, c_R, m_3
$m_3 := (\boldsymbol{ID_I}, \sigma_I, \boldsymbol{cert_I}, proposal_I)$
$MAC_R \leftarrow PRF_{k_{pR}}(ID_R)$
c_I, c_R, m_4
$\sigma_R \leftarrow Sign_{sk_R}(c_I, c_R, m_2, n_I, MAC_R)$
$m_4 := (\boldsymbol{ID_R}, \sigma_R, \boldsymbol{cert_R}, proposal_R)$
$(k'_{eI}, k'_{aI}, k'_{eR}, k'_{aR}) \leftarrow PRF_{k_d}(n_I, n_R)$

cryptographic parameter) and perform the Diffie–Hellman (DH) exchange and random number sharing through the first two messages (M_1 and M_2). Both parties use the current shared DH secret, g^{xy}, and other materials (*Cookie*, n_I, n_R, etc.) to derive seven symmetric keys (k_d, k_{aI}, k_{aR}, k_{eI}, k_{eR}, k_{pI}, k_{pR}). In addition, both parties use their private keys to sign their respective hash values (MAC_I, MAC_R), and send their own identity, public-key certificate, and signature to the other party through M_3 and M_4 to complete signature verification and identity authentication. Generally, after receiving the M_3 (or M_4) message, the communication party comprehensively compares the ID payload information, public key certificate, and preset identity information of the other party (or not set) to verify the legality of the public-key certificate and the identity of the communication party and ensure the security and reliability of the communication.

Because of the various situations involved, hostname verification is a relatively complicated process. In the following subsection, we introduce the knowledge and policy required for matching.

3.2. Contents of X.509 Certificate

X.509 is one of the standard formats for public-key certificates, and it contains the To-Be-Signed (TBS) part, signature algorithm identifier, and actual signature value. The structure is shown in Figure 2. The communicating entity can ask for an X.509 certificate from a trusted certificate-issuing authority. One party in the communication uses the private key to sign the message, and the other party uses the corresponding public key to verify the message. Furthermore, the subject name of the certificate is compared with the user identity to ensure that the identity of the certificate provider is correct. In the entire X.509 framework, there are many RFC definitions and specifications, such as RFC2459 [13], RFC3820 [34], RFC4985 [30], and RFC5280 [5].

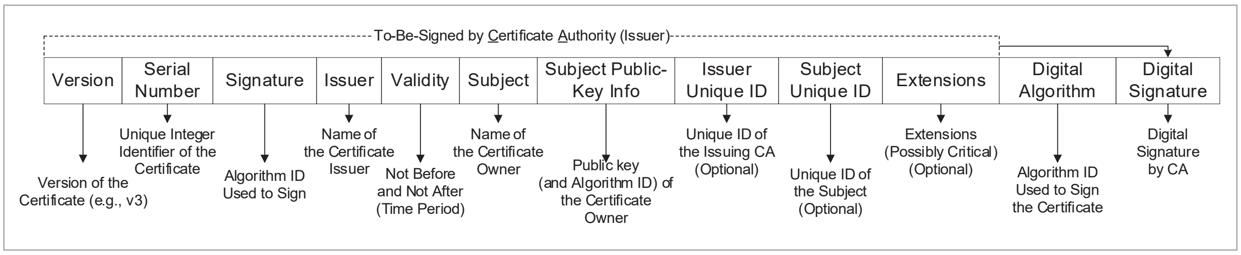
A standard X.509 certificate generally contains three fields: tbsCertificate, signatureAlgorithm, and signatureValue. The tbsCertificate field contains Version, Serial Number, Signature (signature algorithm identifier), Issuer (the entity that signed the certificate), Validity (validity period), Subject (certificate owner), SubjectPublicKeyInfo (certificate owner's public key), and the three optional fields issuerUniqueID, subjectUniqueID, and extensions; the other two fields, signatureAlgorithm and signatureValue, are the algorithm the issuer used to sign the certificate and the actual signature value, respectively.

The common name (CN) and subject alternate name (SAN) fields in the certificate are mainly involved in hostname matching. Both CN and SAN are located in the subject field, but SAN is an optional extension of the certificate. The CN generally contains an FQDN, but it can also contain any ASCII or UTF-8 string with a description of the service. The CN should follow the X520CommonName standard (for example, the substring "CN=" should not be repeated).

The SAN is an extension of the X.509 certificate, and it can be used to store different types of identity infor-

Figure 2

Structure of an X.509 certificate


To-Be-Signed by Certificate Authority (Issuer)
Version
Serial Number
Signature
Issuer
Validity
Subject
Subject Public-Key Info
Issuer Unique ID
Subject Unique ID
Extensions
Digital Algorithm
Digital Signature
Version of the Certificate (e.g., v3)
Unique Integer Identifier of the Certificate
Algorithm ID Used to Sign
Name of the Certificate Issuer
Not Before and Not After (Time Period)
Name of the Certificate Owner
Public key (and Algorithm ID) of the Certificate Owner
Unique ID of the Issuing CA (Optional)
Unique ID of the Subject (Optional)
Extensions (Possibly Critical) (Optional)
Algorithm ID Used to Sign the Certificate
Digital Signature by CA

mation, such as rfc822Name, dNSName, uniformResourceIdentifier, and iPAddress. Each of these types has different restrictions on the format; for example, rfc822Name, dNSName, and uniformResourceIdentifier must be valid IA5String strings (a subset of ASCII strings), and the iPAddress must be encoded in a network byte order.

If the content of the "subject" item in the certificate is empty, the CA must put the SAN extension into the certificate and mark the extension as critical when issuing the certificate. When the "subject" entry in the certificate contains a non-empty DN name, the CA must mark the SAN as non-critical when issuing the certificate. For more specific X.509 naming rules, please refer to RFC4985 [30] and RFC5280 [5].

3.3. Identification Types of IPsec

In this subsection, we summarize the identification types of IPsec according to the relevant RFCs.

The birth and development of IPsec experienced 3 stages: AH and ESP (RFC1825-1829, 1995), IKEv1 (RFC2401-2412, 1998), and IKEv2 (RFC4301-4312, 2005). Later, after the transition of RFC4718 (2006) and RFC5996 (2010), the updated version of IKEv2 (RFC7296) was standardized in 2014, which has better security and accuracy. In addition to these basic RFCs, there are numerous RFCs that supplement specific items, such as RFC2709, RFC3457, RFC4945, RFC5386, RFC6331, RFC7634, and RFC8229.

Given the identification and authentication problems that IPsec may face, RFC4301 [22] defines six IKE ID types: domain name system (DNS), distinguished name (DN), RFC822 (email format), IPv4, IPv6, and key ID. According to different encoding formats, RFC4945 [24] specifically defines 11 different ID identifiers, corresponding to these six ID types. In addition, RFC4595 [25] proposed the ID_FC_NAME format applied to the fiber channel protocol, and RFC7619 [32] proposed the ID_NULL format. With the ID_ANY/ID_NONE type, the 14 ID identifier types are summarized in Table 1.

Table 1
IPsec identification types and their assignment values

ID type	Assignment value
ID_ANY/ID_NONE	0
ID_IPV4_ADDR	1
ID_FQDN	2
ID_USER_FQDN	3
ID_IPV4_ADDR_SUBNET	4
ID_IPV6_ADDR	5
ID_IPV6_ADDR_SUBNET	6
ID_IPV4_ADDR_RANGE	7
ID_IPV6_ADDR_RANGE	8
ID_DER_ASN1_DN	9
ID_DER_ASN1_GN	10
ID_KEY_ID	11
ID_FC_NAME	12
ID_NULL	13

Table 2
Bindings of the identification payload to the contents of end-entity certificates and of identity information to policy in RFC 4945

ID type	Support for send	PKIX Attrib	Cert matching	SPD lookup rules
IP*_ADDR	MUST support	SubjAltName iPAddress	MUST match exactly	Must perform exact matching. Also, MAY perform substring or wildcard matches
FQDN	MUST support	SubjAltName dNSName	MUST match exactly	Must perform exact matching. Also, MAY perform substring or wildcard matches
USER_FQDN	MUST support	SubjAltName rfc822Name	MUST match exactly	Must perform exact matching. Also, MAY perform substring or wildcard matches
IP range	MUST NOT	n/a	n/a	n/a
DN	MUST support	Entire Subject, bitwise compare	MUST match exactly	MUST support lookup on any combination of C, CN, O, or OU
GN	MUST NOT	n/a	n/a	n/a
KEY_ID	MUST NOT	n/a	n/a	n/a

Regarding the use of these identity types, RFC4809 [2] has a corresponding description, which requires that the IPsec implementation must support FQDN and RFC822 (e-mail format, IPv4 address, and IPv6 address. In addition, RFC4945 [24] has made seven clear requirements for the usage of IPsec IDs, that is, the support and matching rules in Table 2 should be met.

3.4. IPsec Hostname Matching Policy

IPsec implementations MUST provide a means for an administrator to require a match between an asserted IKE ID and the subject name or subject alt name in a certificate [22]. Because implementations may use ID as a lookup key to determine which policy to use, all implementations MUST be especially careful to verify the truthfulness of the contents by verifying that they correspond to some keying material demonstrably held by the peer [24]. In addition to domain name strings in the general sense, hostname verification involves IP addresses and emails.

Based on the support and matching situation summarized in Table 2 [24], in this section, we further sort out the different types of ID matching rules and discuss the situation where a certificate contains multiple hostnames.

3.4.1. ID_IPV4_ADDR and ID_IPV6_ADDR

The IPsec implementation must support at least ID_IPV4_ADDR or ID_IPV6_ADDR ID types, and these addresses must be encoded in the network byte order specified in RFC791. In the event of a network address translator (NAT) traversal, the implementation should not fill the ID payload with an IP address. If the other party's IP address is static, and the peer is not behind a NATing device, and the administrator wants to implement this scheme to verify whether the peer source address matches the IP address in the received ID and the IP address in the iPAddress field of the SAN extension of the peer certificate, the user can only consider the IP address as the ID. The implementation must be able to verify that the IP address shown in the ID matches the IP address in the iPAddress field of the SAN extended certificate. By default, implementations must perform this verification. When comparing the content of the ID with the iPAddress field in the SAN extension for equality, a binary comparison must be performed.

3.4.2. ID_FQDN and ID_USER_FQDN

The IPsec implementation must support the two ID types ID_FQDN and ID_USER_FQDN, which provide host-based access control lists for hosts without fixed IP addresses. If the ID contains ID_FQDN, the implementation must be able to verify whether the identity contained in the ID payload matches the identity information contained in the peer entity certificate in the dNSName field of the SAN extension. If the ID contains ID_USER_FQDN, the implementation must be able to verify whether the identity contained in the ID payload matches the identity information contained in the peer entity certificate in the rfc822Name field of the SAN extension. When comparing the content of the ID with the dNSName or rfc822Name field in the SAN extension for equality, a case-insensitive string comparison must be performed. Moreover, the comparison cannot perform substring, wildcard, or regular expression matching.

3.4.3. ID_DER_ASN1_DN

The IPsec implementation must support receiving and generating ID_DER_ASN1_DN types. When generating this type, the implementation must fill in the ID content with the Subject field of the end entity certificate. When this is done, a binary comparison between the two can be successfully performed. If there is no match, it must be treated as an error and the security association setting must be aborted.

In addition, regarding SPD matching, the implementation must be able to perform matching based on a bit-by-bit comparison of the entire DN in the ID and its entry into the SPD. However, it is difficult to use the entire DN in a local configuration, especially in a large-scale deployment. Therefore, the implementation must also be able to perform SPD matching on one or more combinations of the C, CN, O, and OU attributes in the ID DN subject. Implementations may also support substring, wildcard, or regular expression matching on any supported DN attribute from ID (in any combination) to SPD.

3.4.4. Other Types

For the remaining ID types, RFC4945 indicated that ID_IPV4_ADDR_SUBNET, ID_IPV6_ADDR_SUBNET, ID_IPV4_ADDR_RANGE, and ID_IPV6_ADDR_RANGE are still in the experimental stage; ID_DER_ASN1_GN is forbidden to be generated because the recipient does not know how to use it, and ID_KEY_ID is used to specify the pre-shared key for verification and is not in the scope of the certificate verification. Therefore, in this paper, we do not consider these types.

3.4.5. Policy of Multiple Identifications in the IPsec Implementation Certificate

The implementation must support certificates that contain multiple identities. In many cases, in addition to a non-empty subject, the certificate will also include an identity (such as an IP address) in the SAN extension. The implementation shall fill in the ID with the identity that may be named in the peer policy, usually FQDN or USER_FQDN. The recipient must use the identity sent as the first key when choosing a strategy. If there are overlapping strategies caused by wildcards, the receiver must also use the most specific strategy in the database.

4. Model Learning

In this section, we introduce state-machine and model learning and discuss the algorithm selection and other considerations for hostname verification.

The purpose of model learning is to construct a state machine model that describes the operating logic of the target system through the interaction of inputs and outputs. Model learning can be divided into passive learning and active learning. In this study, we employ active learning to initiate a limited number of active queries to the target for inferring a complete target model. In addition, according to the requirements for the expression of the model, we adopt a Mealy machine to describe the DFA model.

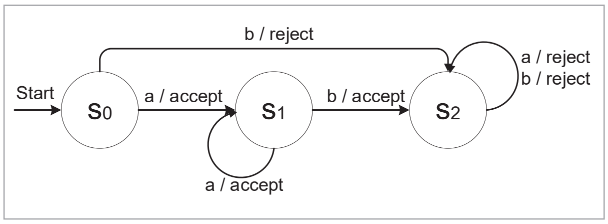
4.1. Mealy Machine

Definition 1. A Mealy machine is a tuple, $M=(I, O, S, s_0, \delta, \lambda)$, where I is a finite set of inputs, O is a finite set of outputs, S is a finite set of states, $s_0 \in S$ is the initial state, δ: $S \times I \rightarrow S$ is a transition function, and λ: $S \times I \rightarrow O$ is an output function.

The Mealy machine can well reflect the characteristics of the DFA model; that is, in any state, a given input only has one state transition and output. Here, we only present the information that facilitates the explanation of the methods employed and results obtained in our study. For more details regarding Mealy machines, please refer to [30].

The Mealy machine and the state graph have a natural corresponding relationship; that is, a graph containing edges and nodes can be used to represent the state transition contained in the Mealy machine. As shown in the basic Mealy machine model in Figure 3, when the model is in the initial state s_0 and receives the input a, the state transition $\delta(s_0, a)=s_1$ occurs, and the output $\lambda(s_0, a)=accept$ is obtained, which corresponds to the edge from s_0 to s_1 and the label *a/accept*. Figure 3 depicts the corresponding collection of strings that can be accepted by the system: {a, ab, aa, aab, aaa, aaab, ...}.

Figure 3
Simple DFA model with alphabet: {a, b}



The output function, λ, can also process multiple consecutive inputs: $\lambda(s,\tau\sigma)=\lambda(s,\tau)\lambda(\delta(s,\tau),\sigma)$ and $\lambda(s, \epsilon)=\epsilon$, where $s \in S$, $\tau \in I$, $\sigma \in I^*$ is a non-empty sequence, and ϵ is an empty sequence. Further, we define the behavior of a Mealy machine M through $A_M(\sigma)=\lambda(s_0, \sigma)$, $\sigma \in I^*$. Thus, we define that two Mealy machines, M and N, are equivalent if and only if $A_M(\sigma)=A_N(\sigma)$ for any $\sigma \in I^*$, denoted as $M \approx N$. Further, we note that $\sigma \in I^*$ is distinguished between M and N if and only if $A_M(\sigma) \neq A_N(\sigma)$.

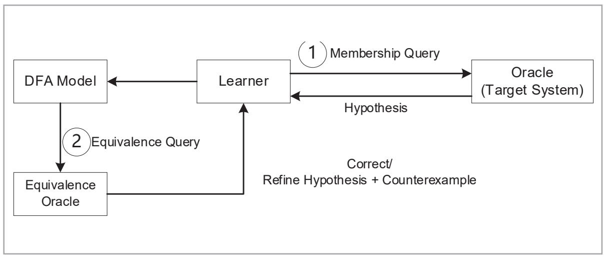
4.2. Model-Learning Framework

We introduce the MAT framework to describe the process and logic of model learning. The MAT framework was proposed by Angluin [1], and it has been widely used in practice. This framework has two major components: the learner and the oracle (target system). The learner wants to perform the inference and modification of the DFA model by constantly querying the oracle, and the oracle has infinite knowledge using which it can answer the learner's queries. As shown in Figure 4, the execution process of the framework is divided into two parts:

1 The first part is membership query: the learning algorithm uses the known input alphabet set I

Figure 4

Model-learning framework



and counterexample set D (D is initially empty); further, it fills in the observation table used to record the query sequence by enquiring the oracle until the table is closed (Different learning algorithms may employ varied recording methods; here, we take the observation table from the L* algorithm as an example for explanation). In this stage, the learner performs multiple rounds of querying. In each querying round, the learner first resets the oracle to the initial state s_0 and then inputs the string sequence $\sigma \in I^*$ to the oracle for querying. The oracle receives the query and processes the response with $A_M(\sigma)$. The learner records the response and completes a round of querying. The learner then builds a model hypothesis H and submits it to the next stage, that is, equivalent query.

2 The other part is equivalence query: at this stage, the learner needs to complete the modification of the model; that is, to verify whether the model hypothesis H is equivalent to the real system ($H \approx M$). If the learner does not search for a distinguishing sequence $\sigma \in I^*$ such that $A_H(\sigma) \neq A_M(\sigma)$ under a certain pre-set search time or range, then the hypothesis is considered equivalent to the real system, and H is output. Otherwise, the learner adds σ to the counterexample set D and returns to the membership query again. Subsequently, the equivalent query is repeated until no new counterexample can be found. At this moment, the learning finishes and we obtain the final DFA model H.

In our tests, the implementation of certificate hostname verification is regarded as our target system. Through the effective interaction with it in the above two steps, the learning algorithm can infer the state machine model of all the hostname-matching results for a certificate in a limited number of queries.

4.3. KV Algorithm and Wp Method

According to the model-learning framework, we need to select the appropriate learning algorithm and equivalent query algorithm to infer the state machine.

The first algorithm used to learn DFA accurately from query models was the L* algorithm proposed by Anguin [1]. Subsequent scholars have designed a variety of other algorithms based on it.

In this study, we used the KV algorithm improved by Kearns et al. [21] as the state machine learning algorithm. Kearns et al. replaced the observation table structure with a discrimination tree, thus achieving higher storage and query efficiency. The storage complexity of the KV algorithm is $O(|\Sigma|n + nm)$, and the time complexity is $O(|\Sigma|n^2 + nlog_m)$, where $|\Sigma|$ represents the size of the alphabet; n is the total number of states in the minimum state model of the target system, and m is the length of the longest counterexample returned by the oracle.

In addition, we used the Wp method [23] as an equivalence query algorithm to perform an equivalent oracle test on the hypotheses constructed by the learning algorithm. In the process of checking, we set a parameter for the test depth. The algorithm takes the test depth plus the number of states of the current model as the upper limit of the check and searches for counterexamples by means of extended tracking. If under the upper limit of the number of states, no counterexample can be found after testing, it can be considered that the state machine hypothesis currently obtained is consistent with the actual DFA model. The Wp method is powerful, but it requires a very large performance overhead; therefore, in actual use, we need to set an appropriate test depth.

4.4. Certificate Template Generation and Alphabet Selection

To test all the different rules in hostname verification, we created 19 certificates with different identifier content to cover specific rules in Section 3.4. For example, a certificate with the common name “CN=*.a.a” can test IPsec’s matching processing of ID_DER_ASN1_DN type IDs and the situation of wildcards. Our template certificate was generated by Libreswan’s certutil library, which can facilitate the generation of certificates with various CN and SAN tags that have different types. For more specific content of the certificate templates, please refer to the appendix A.

In addition, we must create an alphabet that will determine which characters the query sequence of the learning algorithm consists of. Because the performance of the learning algorithm depends on the size of the alphabet, we choose a small set of representative characters to form our alphabet to improve the efficiency of the test. In this study, we used the alphabet set Σ = {a, 1, dot, \s, @, A, =, *, /, -, NULL}, where "dot" represents the character ".", "\s" represents the space character (ASCII 0x20), and "NULL" means zero-byte characters (ASCII 0x00). This alphabet basically covers different types of characters so that we can test possible hostname matching exceptions in the IPsec implementation.

5. Adapting Test for IPsec Implementation

In this section, we introduce the specific test process and analysis results of our implementation of IPsec hostname verification.

5.1. IPsec Implementation

In this study, two popular IPsec implementations with the latest versions, Strongswan 5.9.0 and Libreswan 3.32, were selected for testing. They both implement the corresponding certificate hostname matching module. We need to make appropriate modifications to the source code and recompile it, so that we can call the corresponding implementation of the relevant hostname verification interface for testing.

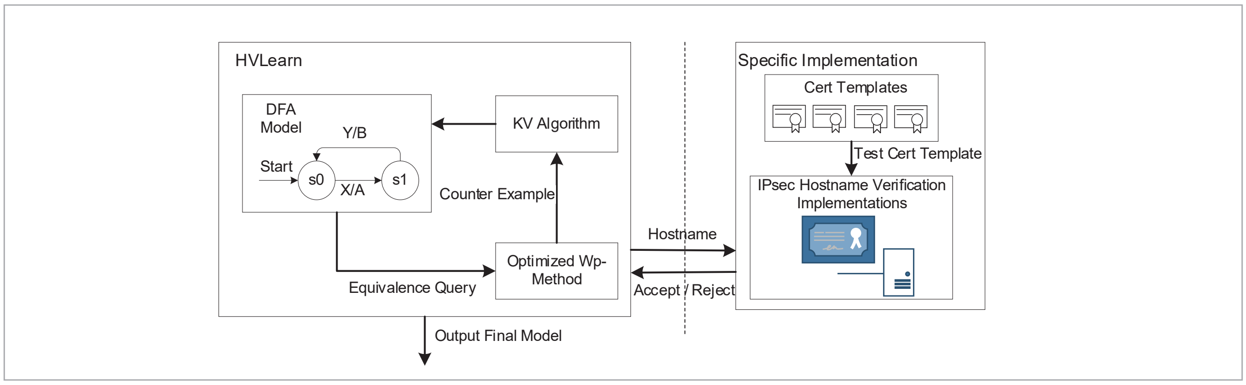
Strongswan is an open-source IPsec implementation, which was originally based on the FreeS/WAN project but has been completely rewritten. It is a complete IP-based VPN solution that supports traditional IKEv1 and the new IKEv2 protocol. The function of Strongswan includes providing multiple authentication methods, such as the X.509 certificate hybrid mode and the pre-shared key for the VPN gateway, among which EAP is also an authentication method (such as EAP SIM based on SIM card or the EAP aka method popular in the mobile environment). In addition, Strongswan also has the advantages of fast VPN connection setting, built-in NAT traversal, dead peer detection, and automatic reduction of the subnet range.

Libreswan is also an open-source IPsec implementation. It is based on the FreeS/WAN code base and extends some additional functions on this basis. It supports most IPsec-related extensions (RFC + IETF draft), including IKEv2, X.509 Certificates, and NAT traversal. By default, Libreswan uses the native Linux IPsec stack (NETKEY/XFRM).

5.2. Model Learning Tool

In this study, we used HVLearn [20] to infer the state machine model. HVLearn is a test tool developed by Sivakorn et al. and is used to analyze the implementation of the SSL/TLS protocol hostname verification. It calls the KV algorithm based on the Learnlib learning library and optimizes the Wp method. In addition, HVLearn provides a convenient Java Native Interface [16] for connecting and testing the implementation of IPsec hostname verification. Figure 5 shows the entire test framework of HVLearn.

Figure 5
HVLearn framework



In each round of model learning, HVLearn will create a new specific string according to the alphabet and the current testing status and input it into the corresponding IPsec certificate verification module (or function) to initiate a query. The certificate verification module processes the input hostname string according to the current certificate template and returns the matching results to HVLearn. After a finite number of queries, HVLearn outputs the corresponding state machine matching model according to all the query results.

To effectively test the implementation of IPsec hostname verification, we adjusted the consistency test depth of the Wp method to 3. To reduce the overhead cost caused by repeated queries, we also used LearnLib's DFALearningCache class to implement the caching of membership query results, check the cache on each new query, and use the cached result when it is found to reduce the cost of repeated queries.

5.3. Analysis Results

By analyzing the inferred state diagram, we found five violations of IPsec hostname verification, which are summarized in Table 3.

Next, we discuss the details of these issues in turn. To facilitate the explanation of specific issues, we simplified the displayed DFA model by reducing the size of the alphabet, leaving only the corresponding key parts. The specific test results are shown in the appendix.

Table 3
RFC violations in the tested IPsec implementations

RFC Violations	RFC	Strongswan	Libreswan
Incorrect handling of space characters in ID_DER_ASN1_DN type IDs	RFC4945	×	×
Incorrect handling of case-insensitive strings in ID_DER_ASN1_DN type IDs	RFC4945	×	√
Incorrect match of wildcards and substring in ID_FQDN type IDs	RFC4945	×	×
Incorrect match of substring in ID_USER_FQDN type IDs	RFC4945	×	√
Unsupported null byte certificate match	RFC7619	×	×

√= OK, ×= RFC violation

5.3.1. Incorrect Handling of Space Characters in ID_DER_ASN1_DN Type IDs

Figure 6 shows the DFA matching results of the two IPsec implementations for the "C=CH, CN=a.a.a" certificate. The double solid circles indicate states that can be matched successfully, and the result of the match is the accumulated character string of the state. As shown in Figure 6 (a), states 0 and 5 indicate that the ID_DER_ASN1_DN type IDs "C=CH,CN=(\s)*(\s)" and "C=CH, CN=(\s)a/A.a/A.a/A(\s)," respectively, can successfully match the certificate containing "C=CH, CN=a.a.a" (\s means that the content of CN is allowed to contain prefixes and suffixes of space characters; a/A indicates that the alphabetic character is not case sensitive).

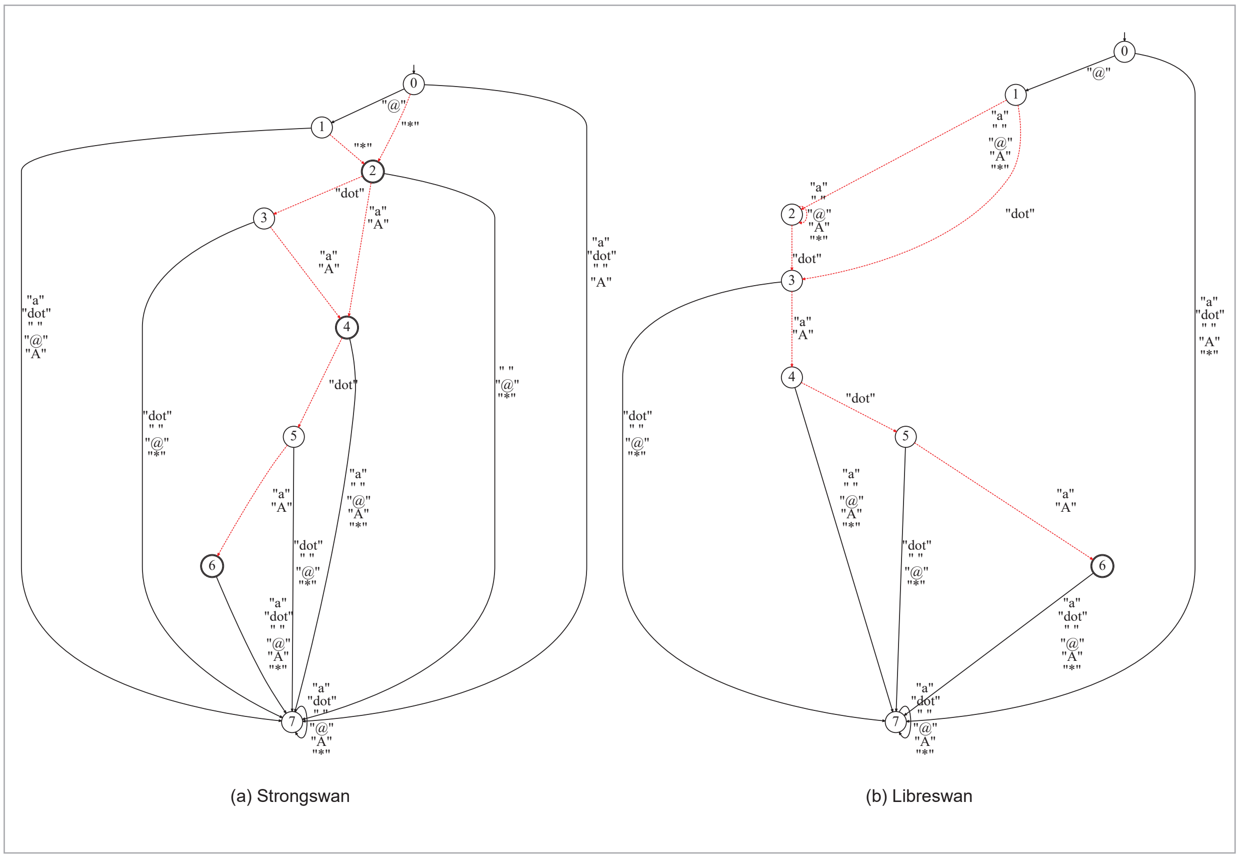
When processing ID_DER_ASN1_DN type ID matching, RFC4945 points out that "the implementation must be able to perform matching based on a bit-by-bit comparison of the entire DN in the ID and its entry in the SPD." According to the experimental results, as shown by the thick blue edges in Figure 6, we found that Strongswan and Libreswan showed a controversial behavior, that is, allowing the content of CN to contain prefixes and suffixes of space characters (in state 0 and state 5). This approach may facilitate the user to configure the implementation, but it will affect the correctness and security of ID matching.

5.3.2. Incorrect Handling of Case-Insensitive Strings in ID_DER_ASN1_DN Type IDs

RFC4945 indicates that strict binary matching must be performed when ID_DER_ASN1_DN type ID matching is performed. However, as shown by the dashed red lines in Figure 6 (a), Strongswan performs case insensitive matching for the certificate containing "C=CH, CN=a.a.a", while Libreswan can correctly perform case sensitive matching, as shown in Figure 6 (b).

Figure 6

IPsec implementation's intersection of DFA in ID_DER_ASN1_DN type ID with cert: C=CH, CN=a.a.a and alphabet: {a, dot, \s, A, *}



5.3.3. Incorrect Matching of Wildcards and Substrings in ID_FQDN Type IDs

According to RFC4945, when processing ID_FQDN type ID matching, substring, wildcard, and regular expression matching MUST NOT be performed for this comparison. However, as shown by the dashed red lines in Figure 7 (a), the matching of ID_FQDN type IDs in Strongswan is obviously much looser, such as the strings "*", "*a/A", "*.a/A", "*a/A.a/A", and "*.a/A.a/A" can be matched with the certificate that contains the "*.a.a" dNSName-type SAN identifier. In addition, as shown by the dashed red lines in Figure 7 (b), Libreswan even has the problem of allowing wildcard certificates to participate in matching, that is, the wildcard (the only character on the leftmost side of the certificate label) can match anything in the leftmost label of the ID.

5.3.4. Incorrect Matching of Substrings in ID_USER_FQDN Type IDs

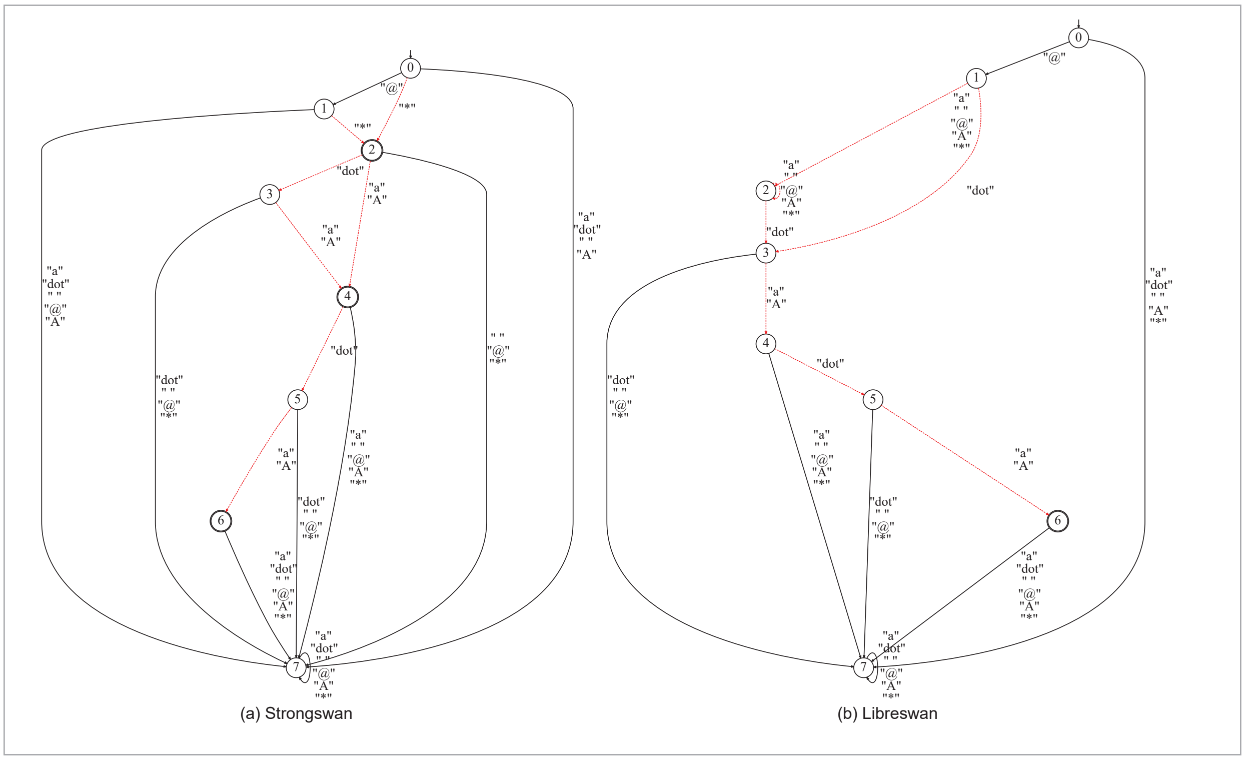
In terms of ID_USER_FQDN type ID matching, RFC4945 indicates that "substring, wildcard, and regular expression matching MUST NOT be performed for this comparison." However, Strongswan allows strings such as "*a/A.a/A", "*a/A", and "*@a/A.a/A" to match a certificate containing the "a@a.a" rfc822 type SAN identifier.

5.3.5. Unsupported Null Byte Certificate Matching

RFC7619 specifies the NULL authentication method and ID_NULL type ID payload and puts forward relevant rules on the use of NULL format IDs. However, for special certificates containing NULL bytes, Strongswan and Libreswan directly end the match by throwing an exception.

Figure 7

IPsec implementation's intersection of DFA in ID_DER_ASN1_DN type ID with cert: CN=abcde and SAN dns:*.a.a and alphabet: {a, dot, \s, @, A, *}



Violations 1)–4) mentioned above may bring the risk of Authentication because attackers can use the corresponding loose character matching to forge their own identity information to bypass hostname verification; the last violation may hinder the certificate verification of ID_NULL type identities.

5.4. Attack Verification

Based on our analysis, we also conducted attack tests to verify that, under specific conditions, the flaws in hostname matching can be used to initiate identity spoofing attacks and man-in-the-middle attacks. The test environment was as follows:

Client and attacker: Ubuntu 18.04 OS + Intel Core I5 8th Gen CPU + 8GB of RAM.

Server: Ubuntu 18.04 OS + Intel Xeon E5-2680 v2 CPU + 32GB of RAM.

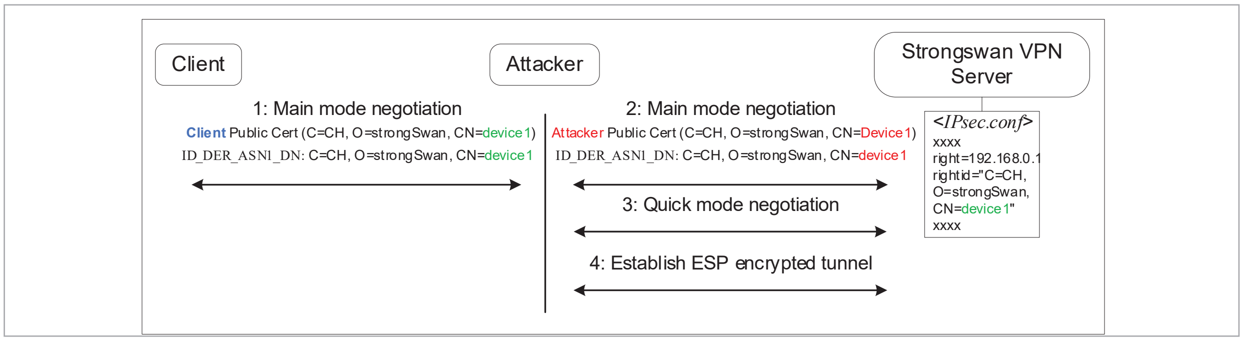
In the following test, it is assumed that the attacker's public key certificate (with available subject content for attack) is issued by a legal CA (usually is private CA in IPsec server) and possesses the corresponding private key, and the attacker can monitor, intercept, and tamper with the messages of the legitimate user. In response to the problems exposed in Strongswan and Libreswan, we give a specific attack scenario to illustrate the effect that can be achieved.

5.4.1. Identity Impersonation Attack Under IKEv1 Protocol Negotiation

In Strongswan, the configuration file (IPsec.conf or Strongswan.conf) provides setting attributes that can be used to verify the identity of the participant: leftid | rightid = <id>, and the specific setting is explained in its wiki [15]. As shown in the attack scenario in Figure 8, the admin sets the server-side configuration file with rightid="C=CH, O=strongSwan, CN=device1", indicating that it only allows users with corresponding field in certificate to access the server device. However, because Strongswan does not distinguish between the upper and lower cases in the CN, when an attacker has a valid certificate issued by a CA with the

Figure 8

Launching an identity Impersonation attack on Strongswan using case-insensitive flaws


Client
Attacker
Strongswan VPN Server
1: Main mode negotiation
Client Public Cert (C=CH, O=strongSwan, CN=device1)
ID_DER_ASN1_DN: C=CH, O=strongSwan, CN=device1
2: Main mode negotiation
Attacker Public Cert (C=CH, O=strongSwan, CN=Device1)
ID_DER_ASN1_DN: C=CH, O=strongSwan, CN=device1
3: Quick mode negotiation
4: Establish ESP encrypted tunnel
<IPsec.conf>
xxxx
right=192.168.0.1
rightid="C=CH, O=strongSwan, CN=device1"
xxxx

subject name "C=CH, O=strongSwan, CN=Device1," an identity impersonation attack can be initiated.

As shown in Figure 8, in IKEv1 signature-based negotiation, the attacker first listens and intercepts the message from the client and impersonates the server to conduct normal main-mode negotiation with the client. After receiving the third message sent by the client, the attacker decrypts and parses the ID and certificate information in it. After judging that the vulnerability can be exploited (recognizing that the client is device1), the attacker initiates a normal IKEv1 main mode negotiation with the true server. When constructing the third message in the main mode, the attacker uses a private key to construct a signature, constructs a forged ID payload (ID_DER_ASN1_DN: C=CH, O=strongSwan, CN=device1), and sends these messages with the public key certificate (with subject name "C=CH, O=strongSwan, CN=Device1") to the server. After verifying the correctness of the signature and certificate chain because the server does not distinguish between upper and lower case characters in the CN, the attacker's certificate will be considered to match the preset hostname, and subsequent negotiation and communication can be successfully completed. At this point, the attacker has completed the impersonation, but the server is completely unaware of it.

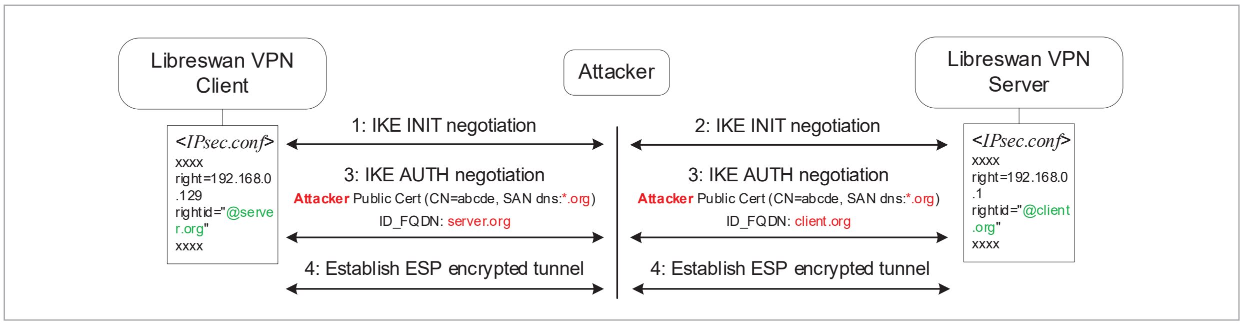
5.4.2. Man-in-the-Middle Attack Under IKEv2 Protocol Negotiation

Similarly, Libreswan can also restrict the identity of communication participants through the rightid = <id> attribute [14]. As shown in Figure 9, the Libreswan client and server have set rightid="@server.org" and rightid="@client.org," indicating that they expect to communicate with the corresponding objects. However, because Libreswan has a flaw in processing ID_FQDN type IDs that allows wildcard matching, when an attacker has a valid wildcard certificate issued by a CA with the SAN "dns: *.a.a," a man-in-the-middle attack can be carried out.

As shown in Figure 9, in the IKEv2 signature-based negotiation, the attacker first listens to and inter-

Figure 9

Launching a man-in-the-middle attack on Libreswan using the flaw that allows wildcard certificate matching



cepts the client's message and pretends to be the server for IKE INIT negotiation with the client. At the same time, the attacker also establishes IKE INIT negotiation with the server. Later, during the IKE AUTH negotiation between the two parties, the attacker uses a private key to construct a signature, constructs a forged ID payload (ID_FQDN: client.org and ID_FQDN: server.org), and sends these messages containing the public key certificate (with SAN label "dns:*. org) to both parties. After receiving the message sent by the adversary, the client and the server decrypt and parse the ID and certificate information, respectively. Because Libreswan has the defect of allowing wildcards in ID_FQDN type ID matching, the client and server will think that the attacker's certificate matches the default hostname, so that the subsequent negotiation and communication can be successfully completed. Therefore, the attacker has completed the man-in-the-middle attack, but the client and server are completely unaware.

6. Conclusions

In this study, we analyzed the problem of hostname verification in IPsec implementation using a model-learning method and inferring the DFA model that can describe the set of all hostnames that match a given certificate subject name. We analyzed two IPsec implementations, Strongswan and Libreswan, and found five violations. Some of these violations may lead to identity impersonation attacks and man-in-the-middle attacks and undermine the security and stability of protocol communication. Because RFC cannot cover all aspects and is not clear in some extreme cases, different IPsec implementations may not be rigorous in handling hostname verification. Therefore, this method can play a role in the development of IPsec implementations and help developers guard against violations and other vulnerabilities that are difficult to find.

In addition, the present study has some limitations: 1) we tested a small number of IPsec implementations and did not conduct a large-scale analysis of IPSec gateway devices; 2) our analysis of the DFA models was performed manually; consequently, the efficiency was low. In the future, we will improve our analysis methods, such as by introducing model checking to analyze state diagrams. Moreover, we aim to analyze the host name verification of more IPSec gateway devices, and even evaluate the implementations of other protocols, to provide methods and technical support for more vulnerability mining scenarios.

Appendix A

In Table 4, we show the detailed test results of hostname verification in Strongswan and Libreswan.

Table 4
IPsec hostname verification matching results using four ID types with different certificate

Certificate	Strongswan				Libreswan			
	ID_DER_ASN1_DN	ID_FQDN	ID_USER_FQDN	ID_IPV4_ADDR	ID_DER_ASN1_DN	ID_FQDN	ID_USER_FQDN	ID_IPV4_ADDR
CN=*.a.a	*; *.a.a	none	none	none	*;*.a.a	none	none	none
CN=a.*.a	*; a.*.a	none	none	none	*; a.*.a	none	none	none
CN=a.a.*	*; a.a.*	none	none	none	*; a.a.*	none	none	none
CN=*.*.a	*; *.*.a	none	none	none	*; *.*.a	none	none	none
CN=.a.a	*; .a.a	none	none	none	*; a.a.a	none	none	none
CN=a.a.a	*; a.a.a	none	none	none	*; a.a.a	none	none	none
C=CH,CN=a.a.a	C=CH,CN=*;C=CH,CN=a/A.a/A.a/A;	none	none	none	C=CH,CN=*;C=CH,CN=a.a.a	none	none	none
CN=a.a.a SAN dns: a.a.a	*; a.a.a	*; a/A.a/A.a/A; *.a/A.a/A; *a/A.a/A	none	none	*; a.a.a	@a/A.a/A.a/A	none	none

Table 4 (continued)

Certificate	Strongswan				Libreswan			
	ID_DER_ASN1_DN	ID_FQDN	ID_USER_FQDN	ID_ IPV4_ ADDR	ID_DER_ ASN1_DN	ID_FQDN	ID_ USER_ FQDN	ID_ IPV4_ ADDR
CN=abcde SAN dns:a.a.a	*	*; a/A.a/A.a/A; *.a/A.a/A; *a/A.a/A	none	none	*	@a/A.a/A.a/A	none	none
CN=a1a SAN dns: a.a.a	*; a1a	*; a/A.a/A.a/A; *.a/A.a/A; *a/A.a/A	none	none	*; a1a	@a/A.a/A.a/A	none	none
CN=a.a.a SAN dns: *.a.a	*; a.a.a	*; *.a/A.a/A; *.a/A; *a/A; *a/A.a/A	none	none	*; a.a.a	@.a/A.a/A; @(Any+).a/ A.a/A	none	none
CN=a.a.a SAN dns: a.*.a	*; a.a.a	*; *.a/A.a/A; **.a/A; *.*.a/A; a/A.*.a/A; *.a/A;	none	none	*; a.a.a	@a/A.*.a/A	none	none
CN=a.a.a SAN rfc822:a@a.a	*; a.a.a	none	(@@)*; @@*a/A; @@*a/A.a/A; (@@)a/A@a/ A.a/A; (@@)*@a/A.a/A	none	*; a.a.a	none	@a/A@a/ A.a/A	none
CN= abcde SAN rfc822:a@a.a	*	none	(@@)*; @@*a/A; @@*a/A.a/A; (@@)a/A@a/ A.a/A; (@@)*@a/A.a/A	none	*	none	@a/A@a/ A.a/A	none
CN= a1a SAN rfc822:a@a.a	*; a1a	none	(@@)*; @@*a/A; @@*a/A.a/A; (@@)a/A@a/ A.a/A; (@@)*@a/A.a/A	none	*; a1a	none	@a/A@a/ A.a/A	none
CN=a.a.a SAN ip:1.1.1.1	*; a.a.a	none	none	1.1.1.1	*; a.a.a	none	none	1.1.1.1
CN= abcde SAN ip:1.1.1.1	*	none	none	1.1.1.1	*	none	none	1.1.1.1
CN= a1a SAN ip:1.1.1.1	*; a1a	none	none	1.1.1.1	*; a1a	none	none	1.1.1.1
CN=NULL	-	-	-	-	-	-	-	-

Note:
Different matching results are separated by ";".
The "a/A" in ".a/A.a/A" means that this character is not case sensitive, that is, the corresponding matching results are a.a, .a.A, .A.a, .A.A.
The "(Any+)" indicates that the matching result allows any string prefix. The "(@@)" indicates that the matching result allows the "@@" string prefix.
Space prefixes and space suffixes are allowed in the CN tag content in Strongswan and Libreswan when ID_DER_ASN1_DN type IDs matching.
A "@" prefix is allowed in the tag content in all ID_FQDN type IDs matching of Strongswan.

Acknowledgments

This work was supported by Innovative Research Groups of the National Natural Science Foundation of China (61521003).

References

1. Angluin, D. Learning Regular Sets from Queries and Counterexamples. Information and computation, 1987, 75(2), 87-106. https://doi.org/10.1016/0890-5401(87)90052-6
2. Bonatti, C., Turner, S., Lebovitz, G. Requirements for an IPsec Certificate Management Profile. The IETF Trust, 2007. https://doi.org/10.17487/rfc4809
3. Chau, S. Y., Yahyazadeh, M., Chowdhury, O., Kate, A., Li, N. Analyzing Semantic Correctness with Symbolic Execution: A Case Study on PKCS# 1 v1. 5 Signature Verification. NDSS, 2019. https://doi.org/10.14722/ndss.2019.23430
4. Cheng, Q., Lu, S., Ma, J. Analysis and Improvement of the Internet-Draft IKEv3 Protocol. International Journal of Communication Systems, 2017, 30(9), e3194. https://doi.org/10.1002/dac.3194
5. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W. RFC 5280: Internet X. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. IETF, May 2008. https://doi.org/10.17487/rfc5280
6. Cremers, C. Key Exchange in IPsec Revisited: Formal Analysis of IKEv1 and IKEv2. European Symposium on Research in Computer Security, 2011. https://doi.org/10.1007/978-3-642-23822-2_18
7. Cui, Y., Yu, T., Hu, J. IKEv2 Protocol Fuzzing Test on Simulated ASA. 2018 IEEE International Conference on Smart Internet of Things (SmartIoT), 2018. https://doi.org/10.1109/SmartIoT.2018.00-16
8. Eronen, P., Hoffman, P. IKEv2 Clarifications and Implementation Guidelines. RFC 4718, October 2006. https://doi.org/10.17487/rfc4718
9. 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. Proceedings of the 2012 ACM conference on Computer and communications security, 2012. https://doi.org/10.1145/2382196.2382205
10. Felsch, D., Grothe, M., Schwenk, J., Czubak, A., Szymanek, M. The Dangers of Key Reuse: Practical Attacks on IPsec {IKE}. 27th {USENIX} Security Symposium ({USENIX} Security 18), 2018.
11. Guo, J., Gu, C., Chen, X., Wei, F. Model Learning and Model Checking of IPSec Implementations for Internet of Things. IEEE Access, 2019, 7, 171322-171332. https://doi.org/10.1109/ACCESS.2019.2956062
12. Harkins, D., Carrell, D. The Internet Key Exchange (IKE)-RFC2409. IETF RFC, 1998. https://doi.org/10.17487/rfc2409
13. Housley, R., Ford, W., Polk, W., Solo, D. RFC 2459: Internet X. 509 Public Key Infrastructure Certificate and CRL Profile. January 1999. Status: PROPOSED STANDARD.9. https://doi.org/10.17487/rfc2459
14. https://libreswan.org/man/ipsec.conf.5.html.
15. https://wiki.strongswan.org/projects/strongswan/wiki/ConnSection.
16. Java Native Interface (JNI). https://docs.oracle.com/javase/8/docs/ technotes/guides/jni/.
17. Kaminsky, D., Patterson, M. L., Sassaman, L. PKI Layer Cake: New Collision Attacks Against the Global X. 509 Infrastructure. International Conference on Financial Cryptography and Data Security, 2010. https://doi.org/10.1007/978-3-642-14577-3_22
18. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 5996, September; 2010. https://doi.org/10.17487/rfc5996
19. Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., T., Kivinen. Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7296, October 2014. https://doi.org/10.17487/rfc7296
20. Kaufman, C. Internet Key Exchange (IKEv2) Protocol. RFC 4306, December 2005. https://doi.org/10.17487/rfc4306
21. Kearns, M. J., Vazirani, U. V., Vazirani, U. An Introduction to Computational Learning Theory. MIT press, 1994. https://doi.org/10.7551/mitpress/3897.001.0001
22. Kent, S., Seo, K. RFC 4301: Security Architecture for the Internet Protocol. 2005. https://doi.org/10.17487/rfc4301

23. Khendek, F., B., Fujiwara, S., Bochmann, G., Khendek, F., Amalou, M., Ghedamsi, A. Test Selection Based on Finite State Models. IEEE Transactions on software engineering, 1991, 17, 591-603. https://doi.org/10.1109/32.87284

24. Korver, B. The Internet IP Security PKI Profile of IKEv1/ISAKMP, IKEv2. and PKIX. RFC 4945, 2007. https://doi.org/10.17487/rfc4945

25. Maino, F., Black, D. Use of IKEv2 in the Fibre Channel Security Association Management Protocol. RFC 4595, July 2006. https://doi.org/10.17487/rfc4595

26. Ninet, T., Legay, A., Maillard, R., Traonouez, L. M., Zendra, O. Model Checking the IKEv2 Protocol Using Spin. 2019 17th International Conference on Privacy, Security and Trust (PST), 2019. https://doi.org/10.1109/PST47121.2019.8949057

27. Ninet, T., Legay, A., Maillard, R., Traonouez, L. M., Zendra, O. The Deviation Attack: A Novel Denial-of-Service Attack Against IKEv2. 2019 18th IEEE International Conference on Trust, Security And Privacy In Computing and Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2019. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00019

28. Nussbaumer, J. Security Analysis for IPsec with EasyCrypt. 2019.

29. Patel, H., Jinwala, D. C. Modeling and Analysis of Internet Key Exchange Protocolv2 and a Proposal for Its Variant. Proceedings of the 6th ACM India Computing Convention, 2013. https://doi.org/10.1145/2522548.2523132

30. Santesson, S. Internet X. 509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name. RFC 4985, August 2007. https://doi.org/10.17487/rfc4985

31. Sivakorn, S., Argyros, G., Pei, K., Keromytis, A. D., Jana, S. HVLearn: Automated Black-Box Analysis of Hostname Verification in SSL/TLS Implementations. 2017 IEEE Symposium on Security and Privacy (SP), 2017. https://doi.org/10.1109/SP.2017.46

32. Smyslov, V., P., Wouters. The NULL Authentication Method in the Internet Key Exchange Protocol Version 2 (IKEv2). RFC 7619, August 2015. https://doi.org/10.17487/RFC7619

33. Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., Khan, L. Smv-Hunter: Large Scale, Automated Detection of ssl/tls Man-in-the-Middle Vulnerabilities in Android Apps. In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14), 2014. https://doi.org/10.14722/ndss.2014.23205

34. Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, M. Internet X. 509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820, 2004. https://doi.org/10.17487/rfc3820

35. Yang, H., Zhang, Y., Hu, Y. P., Liu, Q. X. IKE Vulnerability Discovery Based on Fuzzing. Security and Communication Networks, 2013, 6(7), 889-901. https://doi.org/10.1002/sec.628