Mathematical Problems of Computer Science 53, 49–56, 2020.

UDC 004

# A Solution for Preventing the Rogue Certificate Attack

Sergey E. Abrahamyan[1] and Arman G. Zakaryan[2]

[1]Institute for Informatics and Automation Problems
[2]American University of Armenia
e-mail: sabrahamyan@sci.am, arman_zakaryan20@alumni.aua.am

**Abstract**

In today's online world, internet security heavily relies on the trust in Certificate Authorities. Modern browsers and operating systems provide a comprehensive list to their users, which includes all the CAs they trust by default. This could turn into a serious problem when even one of the CAs is compromised and/or goes rogue. It is especially relevant for enterprise applications, as they are more likely to be targeted for this kind of attack. In this paper, we propose a solution which can mitigate this kind of attack against large organizations. We also discuss the security of the proposed method, offering acceptable security/performance tradeoff.

**Keywords:** HTTPS, TLS, Digital Certificates, Masquerade Attack, Rogue Certificate Attack, Security

## 1. Introduction

Nowadays, security in digital space is of utmost importance. Every day millions of people exchange messages, browse websites, perform banking transactions on the internet, exposing a significant amount of sensitive information to potential adversaries. That is why web service providers implement many techniques to provide sufficient protection to users in the online world. A variety of techniques are implemented to ensure the security of people on the internet, but this paper will primarily focus on authentication. User somehow has to be sure that he is communicating with the intended recipient and not someone who tries to impersonate him/her. For that reason, most of the web services now use digital certificates, which helps users to check if they are communicating with the right entity or not.

Digital certificates are like passports of web services, which leverage the power of public key cryptography to provide a method for server and client authentication. The current standard for authentication technologies on the internet is TLS (Transport Layer Security), a successor of now deprecated SSL (Secure Sockets Layer). Authentication is done by a process called TLS/SSL handshake, during which the client and server exchange a couple of messages (including a signed X.509 certificate sent by the server) and thus verify each others identities.

Before the authentication technology became a security standard on the internet, the communication between the client and server was established by HTTP (HyperText Transfer

Protocol). Data transaction under this protocol is implemented without any encryption. So this protocol is not secure against attacks like man-in-the-middle and eavesdropping. Now that most of the web services use authentication, they utilize the protocol HTTPS, where $S$ signifies that the communication channel is secure. Any data sent through an HTTPS connection is encrypted, and the only two parties that can decrypt ciphertexts are the client and server, who exchanged the encryption key during the handshake phase.

Nowadays, the most common type of certificate used by web services is X.509. It is usually obtained from an entity called Certificate Authority (CA). It contains information about the owner, such as name, certificate version, validity date range, the encryption/signature algorithm supported by the server, owner's public key, and most importantly, the signature of the CA. Certificates are created as follows: web service requests a certificate from CA, by providing them the necessary information, and CAs digitally sign it using their private keys. Thus, whoever gets the signed certificate, can verify its validity by checking the signature on it using CA's public key (which is stored in most modern browsers). After verifying the web service's identity, user generates a pre-master key, encrypts it using tbe server's public key and sends to the server. The server uses its private key to decrypt the pre-master key. Next, the client and server use the same pre-master key to generate a shared secret key and start communicating securely. This system inherently relies on people's trust in the CA and most of the time it pays off. Indeed, it is generally secure against third party intervention, which makes it very practical to use. However, the system is not secure against forged certificates, when the CA itself is involved. For a variety of reasons, such as political, economical or social, CA's may have an incentive to distribute forged certificates to accomplish a masquerade attack, with the purpose of stealing users' private information, spreading misinformation, etc. The probability of such attacks is considered to be very small, as it will diminish the credibility of CA, resulting in lawsuits and financial losses, so most of the modern browsers and OS's don't implement any protection from this kind of attacks. However, if the outcome of the attack is significant, CA's can take the risk and go rogue. Although this can be relevant for individuals, most of the time the targets of this attack are computer networks of big organizations (e.g. banks), as they have access to very sensitive data. The proposed solution could theoretically be extended to individual use as well. In this paper, we consider a masquerade attack, where a malicious CA along with an intruder tries to obtain a legitimate server's identity, using a forged certificate. We also propose a solution, which will prevent such kind of attack.

## 2. Related Work

Various attempts have been made to overcome forged certificate attacks. HTTP Public Key Pinning [1] is one such technique, which protects users from forged certificate attacks to some extent. It is based on trust on first use security model: the first time a user contacts the web service, it gets the certificate and a list of public keys, saying that these are the only public keys, which should be associated with their service in the future. If users receive a signed certificate with a public key that is not present in that list, then it's most probably a rogue certificate and must be rejected. This technique was popular for some time among the major web browsers like Chrome, Firefox and Opera. Around 2018 Chrome announced that they are planning to depreciate and then later remove the support for HPKP in the near future [2] because of multiple security and usability concerns. In the same manner, HTTP strict transport security mechanism offered protection against protocol-downgrade attacks, which are a part of the forged certificate attack family [3]. As a replacement for HPKP, Google announced the Certificate Transparency project [4]. It is a public log of all issued certificates trusted by the browser. Web service owners should regularly check the log for the domain names owned by them, to detect any misissuance of certificates. This provides some level of protection, but it is far from being a near-real time solution. It can take days for website owners to detect any wrongdoings and act appropriately. Another solution is to bind certificates to DNS records. That structure eliminates the need for CA's altogether. This mechanism is known as DANE [5], which is currently used with some websites, but mainly for SMTPS and not HTTPS. The reason it's not widespread is because it requires DNSSec to operate securely, otherwise DNS lookups could be spoofed: man in the middle attacker can simply replace the certificate by another one and successfully carry out an attack. Another scheme for protection, which is highly relevant to our case, is described in [6] Perspectives. In their system, users submit hash values of the certificates they received to a network of notaries, which monitor them continuously. Whenever a user tries to access a web service, it takes the certificate of that service and sends it over to the notary network. There, the certificate is checked with the already stored hash values and in case of a mismatch, notifies the user. This method is a significant improvement in the direction of mitigating the forged certificate attack, however, it has some drawbacks. First of all, it requires the notaries to be a network of trustworthy institutions, such as universities, governmental entities, etc. This really hurts the practicality of the scheme. Secondly, it relies on the history of the submitted certificates. In case when a new certificate is issued for a web service, Perspectives will trigger a false positive warning.

## 3. System Design

First, we fix the formal definition of the attack that our solution aims to mitigate.

### 3.1 Definition of the rogue certificate attack

We assume that the network of the target organization is partially compromised. This can be performed in many ways: attackers could get access to the organization's DNS servers, routers etc. After the compromise, we assume that the attacker can control the traffic and redirect users at any time. Once the traffic is captured and appropriately redirected, attackers initiate the rogue SSL certificate attack. The malicious server sends the user a rogue certificate, signed by the malicious CA. Now the user's browser tries to authenticate the server using the received certificate. In the field "issuer name", the user sees the name of the malicious server. So the user's browser checks the received certificate's signature using the public key of the malicious CA, which is trusted by the browser. In our case, the certificate is signed by the same or any other CA, the root of which is trusted by most browsers. Now, when the user wants to access a web service, he is being redirected to a malicious web page impersonating the real one. As the root of the rogue certificate is a CA, which is trusted by the browser, no warnings will be triggered. The session key that is generated between the client and the web service will be encrypted with the attacker's public key, so he can easily decrypt it and access all the information that is exchanged during the session.

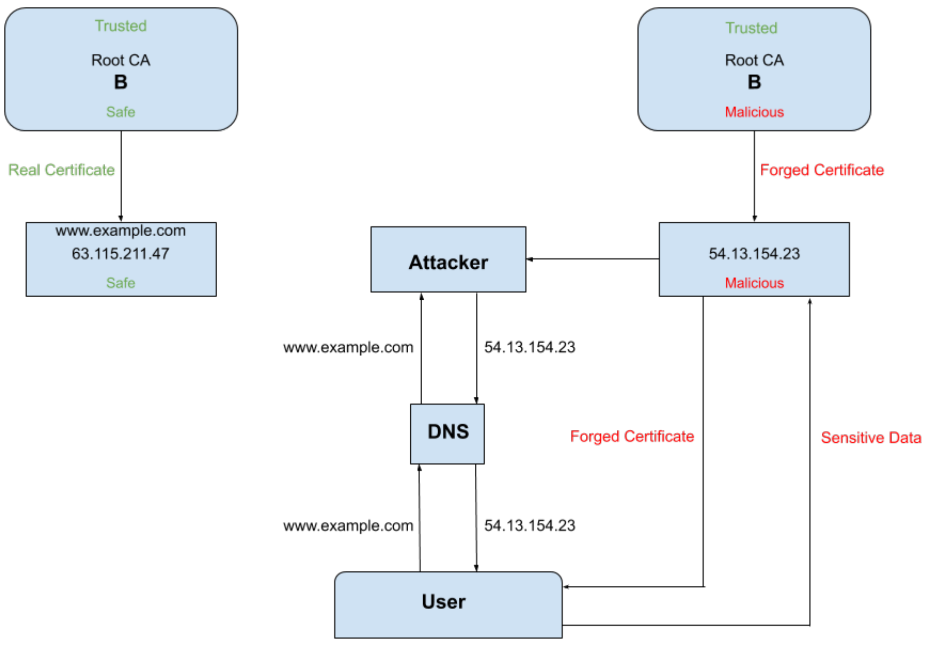

Fig. 1. Attacker takes control of user's DNS and replaces the real IP of the requested website with an IP of a malicious website. This website has a forged certificate from a trusted root CA, so the user accepts it.

## 3.2 Solution Architecture

As a solution, we present a system that could be integrated with all modern browsers. The system has two key roles: local caching and acting as a bridge between local and remote machines.

Caching is needed as a first step local validation of the certificate. To avoid the liabilities of trust-of-first-use approach, if the client is contacting a new web service with no prior mutual history, it naturally skips this first step. After validating it with the remote machine the first time, it stores the hash value of that certificate in a local cache. In the future, as long as the certificate provided by the web server matches the cached one, no further steps are needed. In case of a mismatch, it sends a validation request to the remote server. The remote server is a physical machine that is set up, preferably in a different geographic location (by minimizing the latency/security tradeoff), with reliable internet connection. Inside, the machine contains a number of virtual machines, each using a VPN that reroutes the traffic to various geographic locations. When the remote machine gets a validation request, it checks the certificate with all the virtual machines, and then each VM votes on the consensus. If even one of the machines returns a different hash value of a certificate, the response of the server is negative. In case of a negative reply, depending on the end user's role in the organization, different levels of warnings are issued. If the end user is in a tech savvy sphere, a full description of the issue is presented. Otherwise, the system blocks the connection not letting the user access the potentially malicious resource. As the entire organization is going to use the same remote machine, a caching mechanism is also present there. Similar to the local machine, each validated certificate's hash value is cached in the remote server as well. This way, once a certificate is marked as suspicious, for each subsequent request with that certificate a warning will be issued without all the checks, decreasing the validation time significantly.

The handshake between local and remote machines relies on the prior knowledge about the remote machine. As it is set up by the organization itself, the public key and all the other information typically contained in the certificate are already known and stored on the user's local machine. Even with DNS hijacking, attackers can't impersonate the remote machine, as it would require a remote machine compromise, which is not a part of the considered attack. A proof of concept implementation of the solution is available in our GitHub repository[10].

**Client Side Validation Algorithm:**

**Input:** URL
**Output:** IsSecure

```
1. cert = getSSLCertificateHash(URL)
2. cachedCerts = getLocalCertificatesCache()
3. if cert  cachedCerts:
    (a) cachedCert = cachedCerts.find(cert)
    (b) if cert = cachedCert:
        i. return True
4. isSecure = checkCertificateWithRemoteServer(URL, cert)
5. return isSecure
```

**Central Server Side Validation Algorithm:**

**Input:** URL, cert
**Output:** isSecure

```
1. VMs = getAllAvailableVMs()
2. isSecure = True
3. foreach vm in VMs:
    (a) isSecure = vm.checkCertificateWithVM(URL, cert)
4. return isSecure
```

**VM Side Validation Algorithm:**

```
Input: URL, cert
Output: isSecure
1. localized cert = getSSLCertificateHash(URL)
2. if cert = localized cert:
    (a) return True
3. return False
```

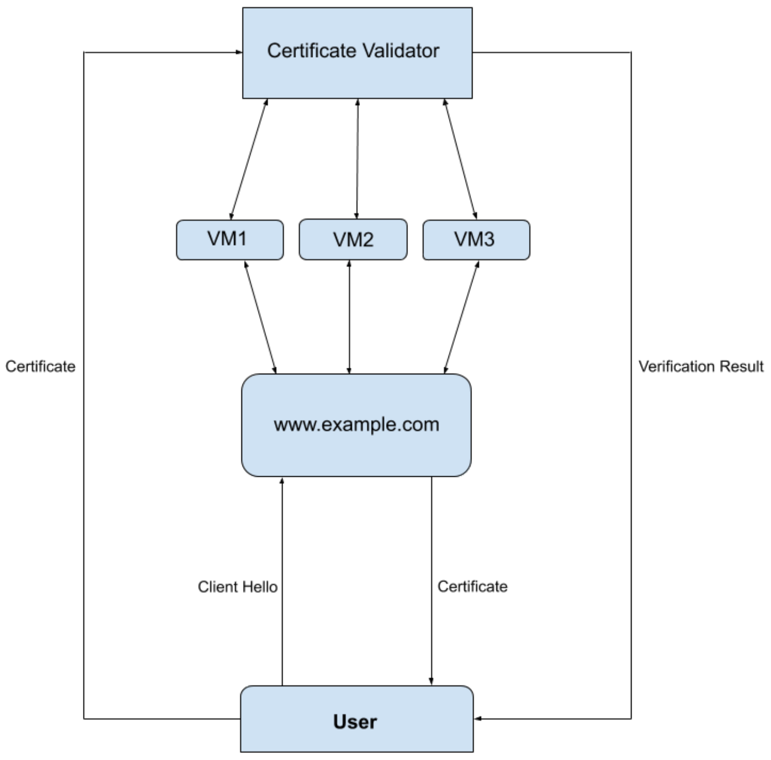

Fig. 2. Server gets the certificate, and sends it to the validator server. Validator server queries all the connected VMs and based on their responses decides to validate or reject the certificate.

## 4. Conclusion

The proposed solution provides significant security overhead over the state of the art solutions. However, the practicality aspect of the solution, in this stage is not that strong.

It can easily be deployed and used by large organizations. Theoretically, the system can be extended to individual use as well, but the maintenance efforts and costs would not be justified for most of the use cases. One direction for further study is extending our approach for individual use as well.

Furthermore, the service will eventually have enough labeled data on authentic/rogue certificates. This can be used to analyze and then further predict the probability of a certificate being rogue. Methods like the one described by Dong et al[7], can benefit from this acquired data, instead of using a generated set of rogue certificates.

## 5. Acknowledgement

The authors would like to thank various sponsors for supporting their research.

## References

[1] C. Evans, C. Palmer and R. Sleevi, *Public Key Pinning Extension for HTTP*, IETF, RFC 7469, doi:10.17487/RFC7469, April 2015.

[2] R. Palmer, "HTTP-Based Public Key Pinning (removed)", retrieved from https://www.chromestatus.com/feature/5903385005916160, 2018.

[3] J.Hodges, C. Jackson and A. Barth, "HSTS Policy", HTTP Strict Transport Security (HSTS), IETF. doi:10.17487/RFC6797. RFC 6797. Retrieved 31 January 2018.

[4] B. Laurie, A. Langley and E. Kasper, "Certificate Transparency", IETF. doi:10.17487/RFC6962. ISSN 2070-1721. RFC 6962, June 2013.

[5] R. Barnes,"DANE: Taking TLS Authentication to the Next Level Using DNSSEC", *IETF Journal*, October 6, 2011, Retrieved August 5, 2018.

[6] D. Wendlandt, D. G. Andersen and A. Perrig, "Perspectives: Improving SSH-style host authentication with multi-path probing", *In Proc. of USENIX'08*, vol. 200, pp. 321-334, 2008.

[7] Z. Dong, K. Kane and J. Camp, "Detection of rogue certificates from trusted certificate authorities using deep neural networks", *ACM Transactions on Privacy and Security*, vol. 19 no. 2, Article no. 5., https://doi.org/10.1145/2975591, September 2016.

[8] D. Fisher, "DigiNotar says its CA infrastructure was compromised", Retrieved from https:// threatpost.com/diginotar-says-its-ca-infrastructure-was-compromised-083011/75594/, 2011.

[9] Z. Durumeric, J. Kasten, M. Bailey and J. Alex Halderman, "Analysis of the HTTPS certificate ecosystem", *In Proc. of IMC'13, ACM*, pp. 291-304, 2013.

[10] Arman Zakaryan and Sergey Abrahamyan, Online. [Available]: https://github.com/armzak1/rogue_certificate_detector

**Submitted 25.03.2020, accepted 18.06.2020.**

# Կեղծ հավաստագրով հարձակումը կանխելու լուծում

Սերգեյ Ե. Աբրահամյան[1] և Արման Գ. Զաքարյան[2]

[1]ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտ
[2] Հայաստանի ամերիկյան համալսարան
e-mail: sabrahamyan@sci.am, arman_zakaryan20@alumni.aua.am

## Ամփոփում

Մերօրյա առցանց աշխարհում համացանցի անվտանգությունը զգալիորեն հիմնված է Հավաստագրային Հեղինակությունների (ՀՀ) հանդեպ վստահության վրա: Ժամանակակից ինտերնետ դիտարկիչները և օպերացիոն համակարգերը օգտատերերին տրամադրում են այն բոլոր ՀՀ-ների ցանկը, որոնք նախապես համարվում են վստահելի: Սա կարող է լուրջ խնդիրներ առաջացնել, երբ վստահված ՀՀ-ներից նույնիսկ մեկը դառնա խարդախության զոհ կամ հենց ինքը դիմի խարդախության: Նման խնդիրների հիմնականում առնչվում են մեծ կազմակերպությունների ծրագրային ցանցերը, քանի որ դրանք են հաճախակի դառնում հարձակումների թիրախ: Այս աշխատանքում մենք առաջարկում ենք լուծում, որը կարող է կանխել նմանատիպ հարձակումները: Անդրադարձ է կատարվել նաև առաջարկվող մեթոդի անվտանգությանը՝ առաջարկելով արագության և անվտանգության ընդունելի փոխհարաբերություն:

**Բանալի բառեր՝** HTTPS, TLS, թվային հավաստագրեր, դիմակահանդեսային հարձակում, կեղծ հավաստագրով հարձակում, անվտանգություն:

# Решение для предотвращения атаки с поддельным сертификатом

Сергей Е. Абраамян[1] и Арман Г. Закарян[2]

[1]Институт проблем информатики и автоматизации НАН РА
[2]Американский университет Армении
e-mail: sabrahamyan@sci.am, arman_zakaryan20@alumni.aua.am

## Аннотация

В современном онлайн-мире интернет-безопасность во многом зависит от доверия к Центрам Сертификации. Современные браузеры и операционные системы предоставляют своим пользователям полный список Центров Сертификации, которым они доверяют по умолчанию. Это может превратиться в серьезную проблему, если хотя бы один из Центров Сертификации будет взломан и/или станет мошенником. Это особенно актуально для корпоративных приложений, так как они с большей вероятностью становятся мишенями для такого вида атак. В этой статье мы предлагаем решение, которое может смягчить данный вид атаки на крупные организации. Мы также обсуждаем безопасность предложенного метода, представляя допустимый компромисс безопасности/эффективности.

**Ключевые слова:** HTTPS, TLS, цифровые сертификаты, маскарадная атака, мошенническая атака сертификата.