Vulnerabilities report

Product: OPC-UA Specifications

19 August 2024

A. Vulnerability #1

Vulnerability identifier*	The “opening of a secure channel” in ECC is more vulnerable to spoofing than its RSA counterpart, because neither the request nor its response identify the receiver.
Editor*	OPC Foundation
Relevant product(s) *	OPC-UA Specification
Relevant version(s) *	1.05
Description *	Unlike in RSA, the Open Channel request and response in ECC do not need to be encrypted, only to be signed, and hence do not mention the receiver’s certificate thumbprint. Therefore, a request can be redirected to another server, and even masqueraded as coming from another client. Yet the opening of the secure channel might succeed, offering the ability for a user to log in and launch commands.
Class of vulnerability	Protocol vulnerability
Impact of the vulnerability	No agreement between the user and the server on the used client machine’s identity. RBAC and logging are impacted. Potential denial of service attacks of redundant servers are worsened.
Access vector	IP Network
Access complexity	Medium
Authentication	High
Confidentiality impact	None
Integrity impact	High
Availability impact	Medium
Indications to trigger the vulnerability*	See detailed explanations
Comments	
CVE request*	No
TLP Level	AMBER

1. Context

The vulnerability described here affects the new version 1.05 of the OPC-UA protocol, that introduces the EC cryptography family, and whose standardization is in progress. Indeed, OPC-UA is currently standardized as IEC 62541 in version 1.04 as of 2017.

When an “Open Secure Channel” request is emitted by a client to a server – or its response from the server to the client – it must contain a security header described in OPC 10000-6 table 51 “Asymmetric algorithm Security header”. The last field of this structure is the *ReceiverCertificateThumbprint*, whose purpose is to indicate what public key was used to encrypt the remaining of the message. But the specification adds “this field shall be null if the message is not encrypted”. With RSA cryptography, the request and the response are always encrypted to protect the confidentiality of the client’s and server’s nonces, that are used to derive symmetric keys. But since in EC cryptography the derivation of symmetric keys uses a Diffie-Hellman key exchange where the client’s and server’s half-keys are only signed, no asymmetric encryption is involved. Hence, according to specifications, there is no *ReceiverCertificateThumbprint* transmitted in ECC, neither in the client’s Open request, nor in the server’s response.

2. Vulnerability Description

Authentication is a fundamental security property in Industrial Control System. We focus on the fact that a server can correctly authenticate requests from a user. This can be stated as follows:

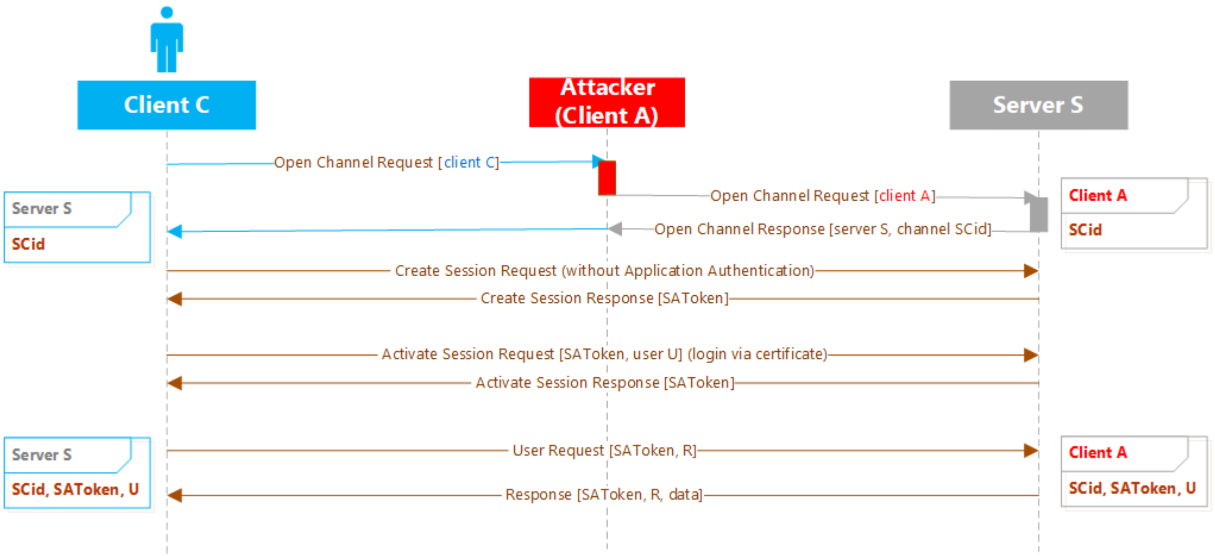
> *« If a server S accepts a request R from a user U on client C, then user U has indeed initiated the request R for Server S on client C »*

We consider the following classical threat model: the attacker controls the network and can intercept, modify, and inject messages; moreover, we suppose that some users, clients, and servers may be compromised, i.e., the attacker has access to their private keys[2]. In this threat model, we slightly refine the above property:

> *« If a server S accepts a request R from an* ***uncompromised*** *user U on client C, then user U has indeed initiated the request R for Server S on client C »*

Indeed, if the client was to be compromised, the attacker could completely spoof the connection and breaking authentication is unavoidable. Within this threat model, we found that the above security property is violated when the new ECC cryptography is used for the secure channel, and no *Application Authentication* is enforced at session activation. We emphasize that the property does hold with RSA cryptography. Therefore, we claim that the introduction of the ECC family as stated in v1.05 strictly degrades the security of the protocol, as it enables a new vulnerability we describe next.

Indeed, as shown in the figure below, the attacker is able to masquerade any user request R from an uncorrupted user U on an uncorrupted client machine C, as coming from U but on a corrupted client machine A.


Client C
Attacker (Client A)
Server S
Open Channel Request [client C]
Open Channel Request [client A]
Open Channel Response [server S, channel SCid]
Server S
SCid
Client A
SCid
Create Session Request (without Application Authentication)
Create Session Response [SAToken]
Activate Session Request [SAToken, user U] (login via certificate)
Activate Session Response [SAToken]
Server S
SCid, SAToken, U
User Request [SAToken, R]
Response [SAToken, R, data]
Client A
SCid, SAToken, U

The attacker can modify the signature, by re-signing with the private signing key of the compromised client A, and hence modify the identity of the sender of a legitimate "Open Secure Channel" request,

[2] This is again a usual hypothesis, e.g. when considering perfect forward secrecy, one of the main advantages of introducing ECC Diffie-Hellman, one supposes a key compromise.

pretending it originates from the corrupted client A. The server will open a secure channel with the corrupted client A[3] and reply. As the server's response does not contain the destination's identity (no *ReceiverCertificateThumbprint* as explained in the *context* section above), it can be forwarded to the legitimate client C that will also open the secure channel on its side. This secure channel can be utilized by the uncompromised Server S and Client C because both share the same symmetric keys, but the server is deceived: the server believes that the connection originates from client A while it actually originates from client C. The server will not realize that received messages originate from client C and not A, until a new authentication of the client machine occurs, through *Application Authentication* at session activation. Using this channel, if a session is created without *Application Authentication* and user U can log in (either with a user certificate or with an unencrypted password in an encrypted channel), then user U's requests from client C, will be accepted by the server as coming from client A. The security property stated above is therefore violated by this attack.

3. Attack scenarios, Impact, and Mitigations

There are at least two main impacts of this attack: on user authorizations and security logging. But there is also a more subtle impact on availability.

Impact on access control and security logs

Indeed, user rights on ICS follow most often a Role Based Access Control scheme that depends on the kind and location of the client machine. For instance, a client control station in the control room may offer more functionalities to a same user as an outside client machine. Hence, the ability of the attacker to masquerade any client, and especially a control station, allows to bypass access control. More directly, the security logging on the server will mention a wrong client machine and this will complexify the understanding and attribution of an attack during forensic analysis, and wrongly raise suspicion about the legitimate and honest user.

Impact on denial of service

Moreover, even without the leak of a client private key, since the *ReceiverCertificateThumbprint* is not present in ECC, the attacker can simply forward an "Open Secure Channel" request addressed to S to another server T. This targeted server T will open a secure channel and reply to the client C, but its "Open Secure Channel" response may or may not be accepted by the client C depending on its security policy: "… a Client shall verify the HostName specified in the Server Certificate is the same as the HostName contained in the endpointUrl. If there is a difference then the Client shall report the difference and may choose to not open the SecureChannel." (cf. OPC 10000-4 §5.5.2.1).

If the client refuses to open a secure channel with the targeted server, this weakness enhances the ability to perform denial of service attacks on the server chosen by the client, while letting the users think of a problem of certificate configuration. On the other hand, the attacker can forward the open request to as many targeted servers as she wishes, and those servers will open a new unused channel for each forwarded request. Indeed, some implementations limit the number of channels to a few dozens. To mitigate the risk of denial of service, the standard recommends to close the oldest unused

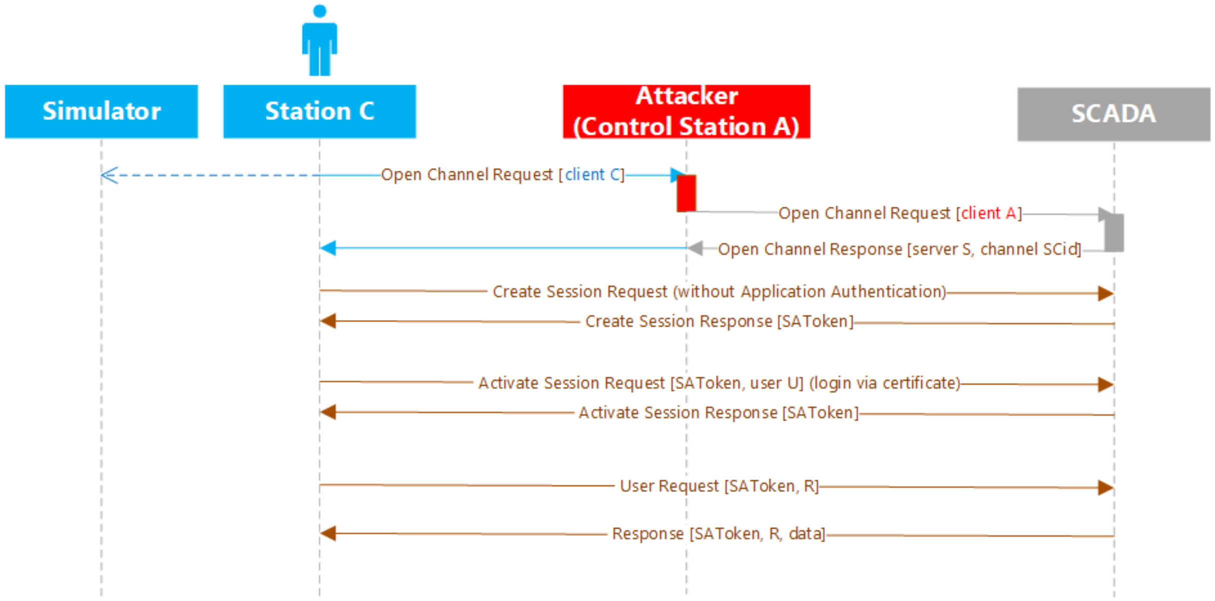
[3] provided that the corrupted certificate is not mentioned in a revocation list.

SecureChannel that has no session assigned before reaching the maximum number of supported SecureChannels (cf. 10000-4 §5.5.2.1).

But if the client supports *Non Transparent Redundancy* of servers as explained in OPC 10000-4 §6.6.2.4, which seems very likely for an ICS, it will probably accept the reply from the targeted server and open a secure channel. That means that, in ECC, an attacker on the network can provoke subtle denial of services of redundant servers, with some of them apparently impossible to reach, and this will have an impact on availability.

An authentication failure scenario

We can combine the two possible attack vectors (modifying both the server identity towards the client and the client's identity towards the server) in the following attack scenario which seems realistic, provided not all security checks are enforced. We suppose an industrial control room with workstations connected to a SCADA server, and a nearby training room with the same kind of workstations but a simulation server. The SCADA server is configured to allow only specific workstations to connect, but all workstations accept a connection to any server. An attacker has corrupted control station A – e.g., just decommissioned from the control room – one of the few allowed by the SCADA. When a legitimate operator (user) U starts its training, the attacker can redirect the open secure channel request coming from the client C in the simulation room to the SCADA in the control room masquerading the workstation A thanks to the vulnerability described in Section 2 (Vulnerability Description), and the assumption that the workstation A is compromised. The SCADA will accept and the attacker will forward its reply to the workstation C in the simulation room, which may accept the reply form the SCADA even if it intended to connect to the simulation server in the first place, because the certificate of the SCADA is allowed and the workstation supports *Non Transparent Redundancy* of servers. The client machine will create a session and the operator will log in with its user certificate (in its badge) and because the server is configured with "*No Application Authentication*", the client private key will not be checked and the session will be activated for him. Now a legitimate operator U on a legitimate client C located in a training room can send a command to the SCADA that will be recognized and logged has being sent from an allowed workstation A in the control room. One can easily imagine the consequences of what was supposed to be a routine review of accident procedures…


Simulator
Station C
Attacker (Control Station A)
SCADA
Open Channel Request [client C]
Open Channel Request [client A]
Open Channel Response [server S, channel SCid]
Create Session Request (without Application Authentication)
Create Session Response [SAToken]
Activate Session Request [SAToken, user U] (login via certificate)
Activate Session Response [SAToken]
User Request [SAToken, R]
Response [SAToken, R, data]

Possible mitigations

First, these attacks can be very easily mitigated by requiring that the *ReceiverCertificateThumbprint* be provided in all cases – RSA or ECC – in the “Open Secure Channel” requests and responses. This mitigation was proved effective on our formal model, that is we were able to establish security proofs with this modification to our model.

Second, enforcing *Application Authentication* at session activation also prevents user login without agreement on the client machine, but not the subtle deny of service of redundant servers.

B. Vulnerability #2

Vulnerability identifier*	Race conditions for user contexts after the handover of a session to a new user.
Editor*	OPC Foundation
Relevant product(s) *	OPC-UA Specification
Relevant version(s) *	All up to 1.05
Description *	When a new user authenticates to an existing session, a request from the old user can be interpreted by the server in the context of the new user, hence abusing its access rights and attributions.
Class of vulnerability	Protocol vulnerability
Impact of the vulnerability	Temporary bypass of access rights.
Access vector	IP Network
Access complexity	Medium
Authentication	Medium
Confidentiality impact	None
Integrity impact	None
Availability impact	None
Indications to trigger the vulnerability*	See detailed explanations
Comments	
CVE request*	No
TLP Level	AMBER

1. Context

The vulnerability described here affects all versions of the OPC-UA protocol. The current standard is IEC 62541 in version 1.04 of 2017, and the future version 1.05 does not address this issue and remains impacted.

In OPC-UA, a user can take over a session from another user, simply by authenticating itself[4]. To do so, an "activate session" request is emitted by the client on behalf of the new user, that the server accepts or refuses. On reception of the server response, the client notifies the new user and, if required, unlocks the local session.

The client can be multithreaded and can emit rapid requests to monitor fast changing values on the server. What happens on the client between the "activate session" request and the server response? The client may continue to send requests, although from a strict security point of view, it would be better to stop all requests, waiting for the server reply. In a careful implementation, the client is able to precisely know in which user context the server response lies, thanks to the integrity provided by the secure channel: before the server reply, the context is the one of the first user, and after a server acceptation reply, the context is the one of the new user.

[4] However, the server can add other requirements, such as the agreement of the owner of the session, or that the rights of the new user encompass those of the current owner.

2. Vulnerability Description

The situation of the server differs from that of the client: after its acceptation of a new user, it cannot know when precisely the incoming requests of the client will be in the new user context because there is no acknowledgment by the client to the server of the server's response. A request of the client on behalf of user 1 can be received after the server has accepted user 2. The only solution offered to the server, in the current standard, for such a request, is to reply in the context of user 2. Moreover, the server should discard all pending responses to user 1 after switching to user 2, and re-evaluate them in the context of user 2. Hence the transition can be brutal and somehow break real time monitoring.

Moreover, this may have bad consequences for Role Based Access Control since user 1 (that could be the anonymous user if authorized by the configuration) could repeatedly launch an action for which she doesn't have the rights, but that would be accepted in the user 2 context, just as user 2 logs in, and would be attributed to user 2.

Indeed, this attack also violates the property stated in the Part A of this report. Note that this violation occurs for a mere network attacker, with no client, server, or user corruption assumption.

3. Impact and Mitigations

This race condition between a request of user 1 and a reply in the context of user 2 could be combined with weaknesses in implementations, and open the way to practical and dramatic attack to temporary circumvent user rights when user 2 logs in.

It is likely that some implementations avoid this break of real time behavior and potential abuse of user rights, and still return responses computed in the context of user 1, even after user 2 has been accepted. To reflect this in the standard and avoid any user confusion, one could specify a smoother transition from user 1 to user 2, by emitting a new session authentication token "SAToken" in the session activation response (cf. OPC 10000-4 §5.6.3.2 table 17); the old session token could still be accepted during a while, until the new one is used by the client, similarly to what is done for the secure channel token, when the symmetric keys are renewed (cf. OPC 10000-6 §6.7.4). This would provide the client acknowledgment of the server approval of a new user. Moreover, this would enable the logging of requests and responses to be coherent between client and server and consistent with user context.

However, this would have an impact on existing implementations because it requires a change in the "activate session" reply message and the management of the SAToken. We however believe this to be a positive change as it corrects an existing blurred behavior. Moreover, thanks to protocol versioning, strict, backward compatibility could be maintained.

C. Vulnerability #3

Vulnerability identifier*	Unexpected weaker cryptographic security for passwords in mode Sign&Encrypt.
Editor*	OPC Foundation
Relevant product(s) *	OPC-UA Specification
Relevant version(s) *	All up to 1.05
Description *	When a password is transmitted for user authentication at session activation, from a client to a server, it is supposed to be encrypted within a dedicated "EncryptedSecret" data container. This is especially the case in mode "Sign". But the specification tolerates the password to be transmitted without an EncryptedSecret data container in an encrypted secure channel (in mode "Sign&Encrypt"). This clearly degrades password security, because in case of a leakage of symmetric channel keys, the password would then be compromised in mode "Sign&Encrypt", while it would have remained secure in mode "Sign".
Class of vulnerability	Security configuration vulnerability
Impact of the vulnerability	Compromising of user credential.
Access vector	IP Network
Access complexity	Medium
Authentication	High
Confidentiality impact	High
Integrity impact	High
Availability impact	None
Indications to trigger the vulnerability*	See detailed explanations
Comments	Enforce a UserTokenPolicy Security Policy also for secure channels in mode "Sign&Encrypt".
CVE request*	No
TLP Level	AMBER

1. Context

The vulnerability described here affects all versions of the OPC-UA protocol. The current standard is IEC 62541 in version 1.04 of 2017, and the future version 1.05 does not address this issue and remains impacted.

In OPC-UA, a client can let a user control a session by providing to the server her login and password in an activation request. This password is normally encrypted within a "EncryptedSecret Data Type" described in OPC 10000-4 §7.41.2.3, except when no security at all is enforced. Otherwise, and especially on secure channels in "Sign" mode, the following means are used:

In case of RSA cryptography, three dedicated symmetric keys are created for the EncryptedSecret: one for encryption, one for symmetric signature (MAC) and one as an initialization vector. The password is encrypted using the encryption key and the IV. The three symmetric keys are encrypted using the server's RSA public key (or a public key of another certificate whose secret key is known by the server) and the whole structure is authenticated with an authentication code using the symmetric signing key. See § 7.41.2.4

In case of ECC, also for the EncryptedSecret, a symmetric encryption key and an IV – used to encrypt the user password – are derived from the established Diffie-Hellman key. The whole structure is signed by the ECC secret key, corresponding to the signing certificate provided by the client. See § 7.41.2.5. Notice that the symmetric half key provided by the server is signed by its private key to avoid spoofing.

The exact algorithms used to protect these encrypted secrets depend on the security policy used for user tokens, that may be different from the security policy of the channel, as stated in OPC 10000-4 §7.41.4, table 193.

2. Vulnerability Description

OPC 10000-4 §7.41.4, table 193, allows a security configuration with no dedicated password encryption, for encrypted secure channels, i.e. channels in mode “Sign&Encrypt” with “Security Policy – Other” and user tokens with “Security Policy – None”. In that case, the security of the user password on the network entirely depends on the confidentiality of the channel symmetric keys. The comment on the row of table 193 that corresponds to this case is “No encryption but encrypted SecureChannel”. Yet, this relaxed configuration may have unexpected consequences.

It is largely accepted that symmetric keys used for encryption and decryption, or to guaranty message integrity with MAC, are more vulnerable than long term keys. Indeed, best practices recommend storing long term keys in a Hardware Security Module (HSM) while ephemeral keys are stored in memory during their lifetime, here during the liftetime of the Security Token of the channel. As a consequence, ephemeral keys are more exposed and subject to compromise than long term keys. For example, the Heartbleed vulnerability on OpenSSL could leak the ephemeral symmetric keys of a TLS connection. Hence an active attacker that records the traffic of an OPC-UA connection, and is able by some means to compromise a set of symmetric keys, can acquire the plain text of the encrypted traffic, and in particular the user password if it has been transmitted in the relaxed security configuration. On the other hand, having the password in an EncryptedSecret container makes a clear difference in that threat model: the attacker needs to recover the symmetric keys derived from long term keys, that do not need to stay in memory for a long time. Indeed, the keys and the password can be erased from the memory by the client as soon as the container is elaborated. Therefore, in the above threat model, an attacker would not be able to compromise the password in mode “Sign”, while it could in mode “Sign&Encrypt”, provided no EncryptedSecret is used, which is permitted in the relaxed configuration.

More generally, we also emphasize that, to simply concatenate the password with the nonce provided by the latest session creation or activation, and encrypting the result, is not a state-of-the-art mechanism for password authentication. More precisely, it provides weaker security guarantees than the use of a password-authenticated key exchange (PAKE) for instance.

3. Mitigations

A proper security configuration – i.e. a valid UserTokenPolicy – should be provided also for channels in mode “Sign&Encrypt”, to better protect user credentials. We recommend to update table 193 to suppress the relaxed case or at least mention that the relaxed case is less secure.

We strongly recommend to change for a mechanism that guarantee the confidentiality of the user password in the presence of a compromised server, and use state-of-the-art password-based authentication protocols such as PAKE.

D. Vulnerability #4

Vulnerability identifier*	A “Signature Oracle” allows a full server impersonation.
Editor*	OPC Foundation
Relevant product(s) *	OPC-UA Specification
Relevant version(s) *	All but especially v1.05
Description *	A server configured with “Security – No Application Authentication” may not verify the certificate transmitted by a client application at session creation, but simply append the client nonce and then sign the resulting bytestring to prove possession of its private key. This can be exploited by a malicious client as a “signature oracle” to produce valid server's signatures that are then sent to an honest client in OpenChannel and CreateSession responses, allowing to impersonate the honest server. From there, request confidentiality, password confidentiality, and user authentication towards (honest) servers are falsified. In ECC, this can be exploited to mount a full server impersonation. In RSA, this allows an authentication violation after establishing a channel, as well as a so-called Unknown Key Share (UKS) attack. Using the same “signature oracle” vulnerability, a similar full server impersonation attack affects a connection with OPC-UA on HTTPS, if the TLS channel does not use application certificates.
Class of vulnerability	Protocol vulnerability
Impact of the vulnerability	Breaks server authentication, password secrecy, confidentiality, integrity and user authentication of requests.
Access vector	IP Network
Access complexity	Medium
Authentication	High
Confidentiality impact	High
Integrity impact	High
Availability impact	None
Indications to trigger the vulnerability*	See detailed explanations
Comments	
CVE request*	No, but CWE request.
TLP Level	AMBER

1. Context

The vulnerability described here affects all versions of the OPC-UA protocol including the version 1.04 currently standardized as IEC 62541. Nevertheless, the impact is more severe on the new version 1.05 of the OPC-UA protocol, that introduces the EC cryptography family, and whose standardization is in progress.

Some OPC-UA servers support the conformance unit **Security – No Application Authentication**, that allows to configure a server to not check the client certificate sent in in the “Create Session” request. It is our understanding that this option has been introduced in OPC-UA to support the HTTPS profile, where – as it is often the case in IT – only the server certificate is checked by the client. The flaw with this construction is that it allows the server to sign the challenge proposed by the client, to prove its possession of its application certificate, without either (i) verifying its content or (ii) adding context to the signature payload. This results in a signature oracle, where an attacker controlling a client accepted by the server can make the server sign anything to reused elsewhere in the protocol.

This signature oracle may be used by an attacker, to impersonate a server S in face of any honest client C, provided she has access to a corrupted client C_A. We only need to assume that S implements the conformance unit Security "No Application Authentication" and that S will accept to open a session in this mode for client C_A. In particular, the attack still works on a client C that does not accept this mode, or a server S that also accepts other modes.

We emphasize that servers are likely to be configured this way (accepting both kind of security modes) in order to be compatible with HTTPS profiles.

2. Vulnerability Description

2.1. The signature oracle

OPC 10000-4 §5.6.2.2 table 15, specifies the signature that must be present in the response to a “Create Session” request: *this is a signature generated with the private key associated with the server Certificate. This parameter is calculated by appending the client Nonce to the client Certificate and signing the resulting sequence of bytes. If the client Certificate contains a chain, the signature calculation shall be done only with the leaf Certificate. For backward compatibility a Client shall check the signature with the full chain if the check with the leaf Certificate fails. The Signature Algorithm shall be the Asymmetric Signature Algorithm specified in the Security Policy for the Endpoint. The Signature Data type is defined in 7.37.*

Therefore, the payload that is signed neither contains any description of what has been signed, nor the current context. This contradicts the usual good practices and yields a signature oracle, as we shall see.

OPC 10000-7 §5.3 table 11 authorizes by the conformance unit ***Security – No Application Authentication*** to configure a server to not check the client certificates given in the “Create Session” request: *The Server supports being able to be configured for no application authentication, just User authentication and normal encryption/signing: Configure Server to accept all certificates. Certificates are just used for message security (signing and encryption). Users level is used for authentication.*

But even in that configuration, the client application certificate and nonce are still transmitted by the client, and then concatenated and signed by the server, because the client must still check the signature, to be sure that the server possesses the private key associated to its certificate.

OPC 10000-4 §5.6.2.2 table 15 also specifies that the client nonce, present in the “Create Session” request, must have a minimal size, but no maximal size is prescribed: *client Nonce: ByteString. A random number that should never be used in any other request. This number shall have a minimum length of 32 bytes. Profiles may increase the required length. The Server shall use this value to prove possession of its Application Instance Certificate in the response.*

Therefore, an OPC-UA server configured with the conformance unit **Security – No Application Authentication** offers to its client a signature oracle: it suffices for the client to provide two bytestrings of minimum length in a create session request, to obtain in the server response, the signature of the concatenated bytestrings, made with the private key of the server application certificate. This works because the first bytestring, supposed to be a valid CA certificate, is not checked in this relaxed configuration. Hence an attacker, controlling only a corrupted client accepted by this server to open a channel and create a session – without checking client certificate as the conformance unit No Application Authentication allows it – is able to obtain the signature oracle.

2.2. Assumptions

We stress that the signature oracle prerequisites should be considered plausible:

- Allowing the conformance unit "No Application Authentication" (among others) allows the server to be compatible with a lot more types of clients, typically those using the HTTPS channel mode. This shouldn’t be rare if the server offers a connection in OPC-UA over HTTPS, but might also be the case with some implementations of OPC-UA Binary, favoring interoperability.
- There is at least one malicious or compromised client. The attacked server accepts to open a channel and create a session with this client. This is also a reasonable assumption: considering compromised clients is precisely what motivates the security layer of OPC-UA.
- The signature oracle is at session creation, and not at session activation, hence it does not require any user information (no need of user's credential or password).

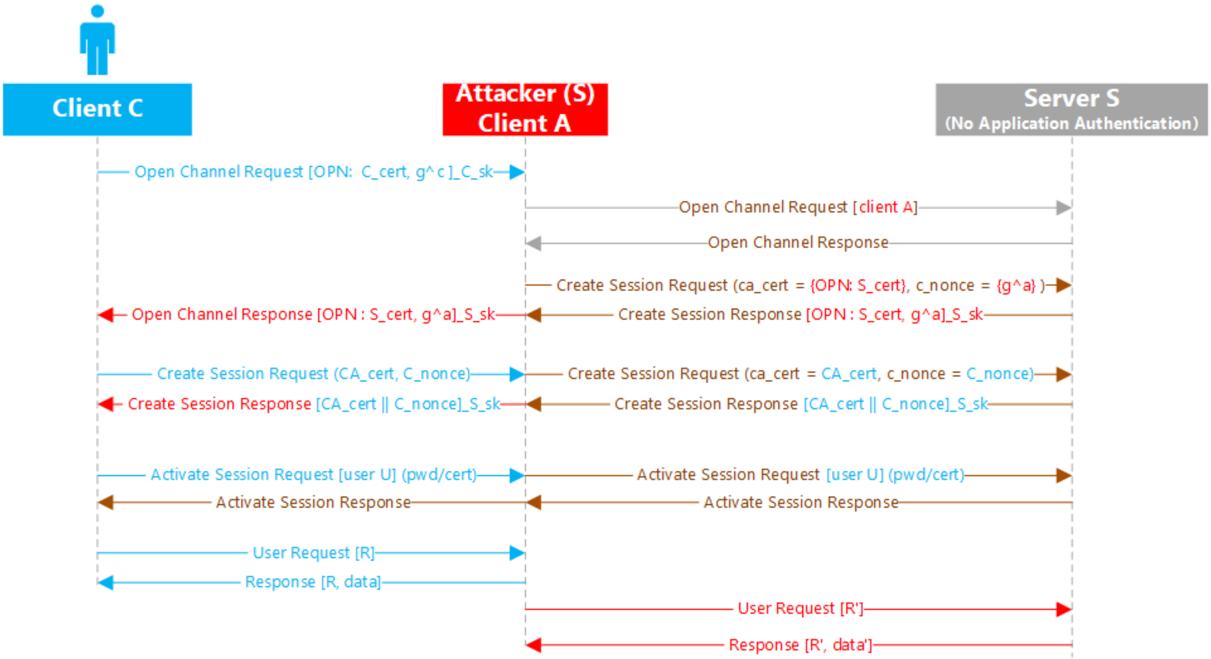
2.3. Exploiting the signature oracle to violate server authentication

We now describe how the signature oracle allows an attacker to impersonate the server during an “Open Channel” response. This attack works in both versions 1.04 (RSA) and 1.05 (ECC). For the sake of clarity, we detail the different steps for version 1.05 (ECC). A very similar attack also affects version 1.04 (RSA).

Consider an honest client C, a server S, and an attacker in control of a compromised or malicious client C_A. The attacker is willing to impersonate the server S at the honest client C. We now describe the attack steps, which are also depicted in the 5 first arrows of the figure below:

- The client first sends an “Open Channel” request, intended for Server S. The attacker intercepts this message and must now respond with an “Open Channel” response. In OPC-UA Binary, an “Open Channel” response from a server to a client must be signed to be authenticated by the client. Therefore, the attacker should obtain a signature over the malicious response it can craft with its own ECC share g^a.

- To obtain this signature, the attacker simultaneously (or previously) opens a channel between client C_A and server S. Once this is done (possibly in advance to save time), the attacker-controlled client C_A exploits the signature oracle by sending a “Create Session” request to the server, as explained in Section 2.1.
- The attacker can thus send a signed “Open Channel” response to the client C, which can validate the response coming allegedly from server S, while it originates from the attacker.


Client C
Attacker (S)
Client A
Server S
(No Application Authentication)
Open Channel Request [OPN: C_cert, g^c]_C_sk
Open Channel Request [client A]
Open Channel Response
Create Session Request (ca_cert = {OPN: S_cert}, c_nonce = {g^a})
Open Channel Response [OPN : S_cert, g^a]_S_sk
Create Session Response [OPN : S_cert, g^a]_S_sk
Create Session Request (CA_cert, C_nonce)
Create Session Request (ca_cert = CA_cert, c_nonce = C_nonce)
Create Session Response [CA_cert || C_nonce]_S_sk
Create Session Response [CA_cert || C_nonce]_S_sk
Activate Session Request [user U] (pwd/cert)
Activate Session Request [user U] (pwd/cert)
Activate Session Response
Activate Session Response
User Request [R]
Response [R, data]
User Request [R']
Response [R', data']

This attack in itself violates a crucial security goal in versions 1.04 and 1.05: server authentication at the end of the Secure Channel sub-protocol.

2.4. Full server impersonation in version 1.05

With EC cryptography, the derivation of symmetric keys uses a Diffie-Hellman key exchange, where the client’s and server’s half-keys are only signed, and no asymmetric encryption is involved. Hence, according to specifications, an attacker able to spoof a server signature can impersonate the server and open a secure channel with a legitimate client. This breaks the server authentication by the client at channel establishment.

The attack goes on when the client sends a “Create Session” request, transmitting its application certificate to be verified and a nonce. Indeed, the attacker simply needs to forward the challenge to the attacker-controlled client C_A, who will again exploit the signature oracle with a new "Create Session" request and will receive in return the signature from the server. Once the legitimate client C is deceived by the attacker who impersonates the server S, it may activate the session by a user authentication, either with login and password – compromising the user password because there is no challenge response – or by a user certificate. In both cases, user activation allows the attacker to abuse of user rights, compromising confidentiality and integrity of all user requests and server responses. The attacker is later able to impersonate any user who attempts to logs in at client C, even when credentials are used (since the challenge-response credential-based mechanism is not bound to the identities of the client and server, so challenges from a server S sent to the attacker can be forwarded to the attacked client C, provided a careful handling of server nonce).

2.5. The case of version 1.04

Nevertheless, in RSA cryptography (used for all secure channels in v1.04), the server signature oracle allows the attacker to forge an acceptable response to an “Open Channel” request from a legitimate client, breaking formally the authentication of the server by the client. But this attack has no further practical consequences, because the client request, that contains the client’s nonce, is encrypted with the server public key, and the attacker cannot read it, and hence cannot derive the symmetric keys. So, the channel just established, is not usable by the attacker. Yet, server authentication is formally violated.

This attack is also exploitable in the OPC-UA HTTPS profile, that rely on a TLS channel without Application certificates, because in that configuration, Application Authentication with OPC-UA certificates is not done during TLS channel establishment, but at session creation and activation. So once the TLS channel between the legitimate client C and the attacker is opened (the attacker must own a simple TLS server certificate, there is no need of the signature oracle for that), and another TLS channel is established between the attacker-controlled client C_A and the server S, the same masquerade of the targeted OPC-UA application server can start at session creation, using the signature oracle, and the same Man-in-the-Middle attack with abuse of user authentication will take place.

3. Impacts and Mitigations

To summarize, we assess that, from the point of view of the legitimate client using OPC-UA Binary, authentication of server by client is violated at the end of the Secure Channel subprotocol in both versions 1.04 and 1.05.

Much more sever is the attack for version 1.05 when EC cryptography is used for Secure Channels, as explained in Section 2.4, and for version 1.04 when a TLS channel without Application certificates is used, as explained in section 2.5. These attacks violate authentication of server by client, message integrity and confidentiality at any step of the protocol (Create Session, Activate Session, User Request, User Response). In particular, user request confidentiality and integrity is violated. This also means that a full server impersonation is possible. We also show in Section 2.4 how user authentication and password confidentiality are violated. All of those attacks are valid even if the legitimate client makes a connection on an endpoint with full security enabled (i.e. without the "No Application Authentication" relaxed configuration).

This vulnerability can be fixed by providing a context to the signature done by the server in response to the client's challenge, following best practices for signatures: explicitly defines what is signed and for whom. Otherwise, the conformance unit **Security – No Application Authentication** should be forbidden, especially with Secure Channels in ECC, but also with TLS channels in RSA or ECC.

4. Supplements

There are at least two other signatures of the same kind in OPC-UA:

- in “Activate Session” requests, both for a new user or to switch to a new secure channel, the client signs the server application certificate concatenated with the server nonce, with the private key associated to its application certificate;

- in “Activate Session” requests, also, if the user authentication is based on a user certificate, then the same byte-string (SA certificate || server nonce) is signed by the private key associated with the user certificate.

But since the first of the two byte-strings, the server application certificate, must always be checked by the client, these two cases are not signature oracles.

Another best practice is to avoid using the same RSA key for encrypting and signing (even if this is not the flaw we exploit here).

E. Vulnerability #5

Vulnerability identifier*	A KCI attack allows a user impersonation.
Editor*	OPC Foundation
Relevant product(s) *	OPC-UA Specification
Relevant version(s) *	All
Description *	The compromise of the private key associated to the application certificate of an OPC-UA server allows an attacker to impersonate an uncompromised user (who is logged on an uncompromised client) to the server. While key compromise allows to trivially impersonate the compromised party, the attack described here does the opposite, i.e., it allows to impersonate an uncompromised entity (here a user) to a compromised one (here a server). Such attacks are known as Key Compromise Impersonation (KCI) attacks. Indeed, one might expect certificate-based user authentication on a client machine not be impacted by the leak the server's keys. (We note that password-based authentication is broken as soon as a user connects to a server whose keys have been compromised.) In summary the KCI attack allows for the following: when the server long-term keys are leaked an attacker can impersonate an honest user (who authenticates with her certificate) to the server and send arbitrary user requests.
Class of vulnerability	Protocol vulnerability
Impact of the vulnerability	Breaks user authentication of requests.
Access vector	IP Network
Access complexity	Medium
Authentication	High
Confidentiality impact	High
Integrity impact	High
Availability impact	None
Indications to trigger the vulnerability*	See detailed explanations
Comments	
CVE request*	
TLP Level	AMBER

1. Context

The vulnerability described here affects all versions of the OPC-UA protocol including the version 1.04 currently standardized as IEC 62541. We consider a threat model where machines are supposed free of hardware or software defects, assuming high cybersecurity maturity of implementations. However, we suppose that it is possible to compromise private keys, for instance by attacking the PKI, side channel or insider attacks. If the attacker learns the private key of an OPC-UA application certificate, we say that this certificate, and by extension the agent, is compromised.

Hence, when a server is compromised the attacker can impersonate it. However, it should not have access to authentication credentials, and should not be able to use service or user accounts. Moreover, if client or even server machines are compromised, the impact on process control should be limited to what is allowed to the anonymous user e.g., the visualization of some part of the process.

2. Vulnerability Description

We now present the details of the user authentication spoofing attack. We emphasize that the main requirement for the attack is the compromise of a server's keys. The attack also requires the compromise of a client machine's keys which should be easier.

Recall that a user authenticates at session activation, by sending the certificate as an identifier of the user, and adding a signature of the server certificate and the latest server nonce to prove possession of the associated private key. This signature can be performed by a smartcard to better protect the private key. Note in particular that the client is not identified by the signature.

When an uncompromised user U on an uncompromised client machine C decides to log in a server S that has been compromised, the attacker will intercept the messages and impersonate the server during channel opening and session creation. At the same time (or before), the attacker creates a session on the server, using a compromised client certificate C_A. When the real server S provides a fresh server nonce to the compromised client C_A in the create session reply, the attacker will forward it in its create session reply to client C. Hence, clients C_A and C share the same server nonce, for their respective sessions. Now when the user U will sign the concatenation of the server certificate and the server nonce, on the uncompromised client C, and send it to the attacker impersonating server S, this signature can be used by client C_A to spoof the authentication of U on the real server S.

The attack involves an uncompromised client C, an uncompromised user U and a server S whose keys have been compromised. We also suppose that the keys of another client C_A have been compromised. In more details, the steps of the attack are as follows:

1. C sends an Open Channel Request to S. The attacker intercepts it. Knowing S's long-term keys, the attacker forges a valid Response. Hence the attacker also knows the channel keys.
2. C sends a Create Session Request to S. The attacker intercepts it.
3. The attacker opens a channel between C_A and S and creates a session se_1 between C_A and S. At session creation the server generates the nonce S_Nonce.
4. Knowing S's long-term keys and the channel keys, the attacker forges a valid Response to C's session creation request (step 2), using the same server nonce S_Nonce as above. C records this session se_2 in its local state.
5. User U on client C sends an activation request for se_2 to S. The attacker intercepts it. Knowing S's long-term keys and the channel keys, the attacker extracts the signature sign(< S_cert, S_nonce >, U_sk) created by U to authenticate.
6. The attacker activates session se_1 using certificate U_cert and U's signature sign(< S_cert, S_nonce >, U_sk), where U_sk is the private key associated to U_cert. As the server S has created S_Nonce for session se_1, this session creation is accepted.
7. The attacker can now send arbitrary user requests in se_1 without any further intervention from U.

Impact and Mitigation

The compromise of an OPC-UA SCADA server's private key, provides the ability to hijack the session of a legitimate user on an uncompromised client machine, allowing the attacker to send arbitrary requests on behalf of the user, even if the latter is uncompromised and uses state-of-the-art authentication on a smart card.

To summarize, we assess that, from the point of view of the server, the authentication of an uncompromised user is violated as soon as the server and a third-party client are compromised. This type of attack, where the compromise of a server key allows to impersonate an uncompromised user to the server is known as a Key Compromise Impersonation (KCI) attack. TLS 1.2 suffered from such an attack (see https://kcitls.org/). Modern protocols are designed to resist this kind of attacks, see for instance the TLS 1.3 specification [RFC 8446, Appendix E.1] that lists resistance against KCI attacks as an explicit goal.

This vulnerability could be fixed by adding information about the client to the user's signature. For instance, adding the hash of the client public key prevents the attack i.e., the signature sign(< S_cert, S_nonce, C_pk >, U_sk), where C_pk is the public key of the client's certificate. This avoids the attack when the user authenticates on an uncompromised client.