CLAROTY

TEAM82

Exploiting OPC-UA in Every Possible Way:

Practical Attacks Against Modern OPC-UA Architectures

Sharon Brizinov, Noam Moshe @ Claroty Research - Team82

$whoami



Sharon Brizinov

Vulnerability researcher - CTFs, Pwn2Own, DEFCON blackbadge, mostly breaking PLCs



Noam Moshe

Vulnerability researcher - Pwn2Own, mostly breaking IoT clouds

* Special thanks to Claroty Team82 researchers: Uri Katz, Vera Mens



Background

Researched dozens of OPC-UA protocol stacks and products

Found core issues in protocol implementations

~50 CVEs: DoS, Info leaks, RCE

~12 unique generic attacks

Open-source tools

- OPC-UA fuzzer
- OPC-UA exploitation framework

Three Pwn2Own ICS ~$200k 💰 💰 💰

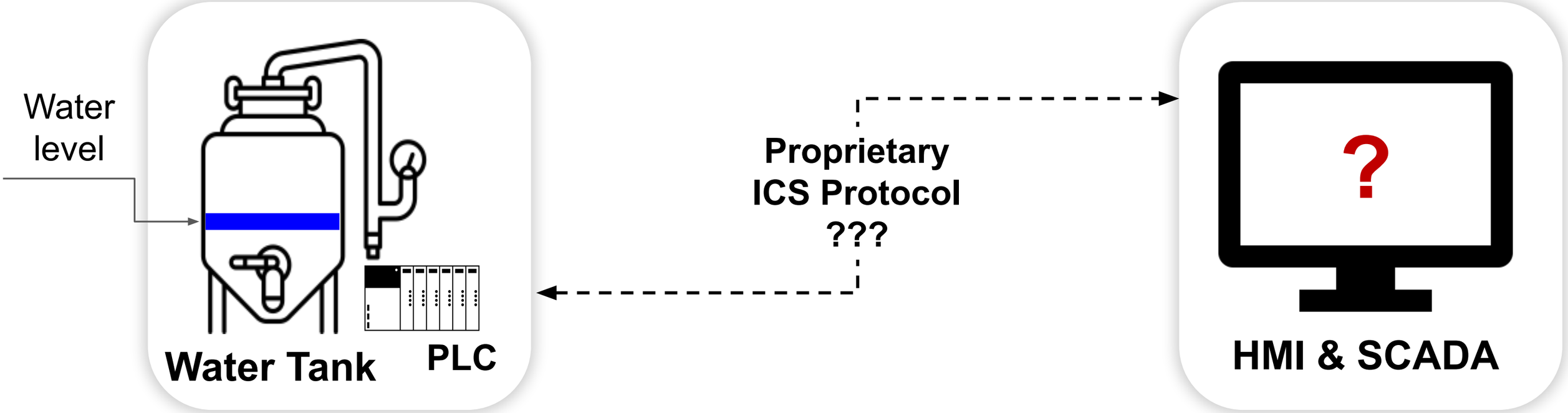
Stack/Application Name	Lang	Complex Deep Nested Variants DoS	Worker Starvation DoS	Long Chunks DoS	Unlimited Monitored Items DoS	UTF8 - UTF16 Conversions
node-opcua	NodeJS	V	V	CVE-2022-21208	CVE-2022-24375	V
open62541	C	V	V	CVE-2022-25761	V	V
freeopcua (c++)	C++	V	V	V	CVE-2022-24298	V
python-opcua	Python	V	V	CVE-2022-25304	V	V
opcua-asyncio	Python	V	V	CVE-2022-25304	V	V
eclipse-milo	Java	V	V	V	CVE-2022-25897	V
ASNeG OpcUaStack	C++	V	V	CVE-2022-24381	V	V
locka99	Rust	CVE-2022-25903	V	CVE-2022-25888	V	V
Unified Automation	C++	V	V	V	Fixed, No CVE	V
OPC Foundation .NET Stack	C#	CVE-2021-27432 (*	V	CVE-2022-29864	V	V
Softing OPC UA SDK	C++	V	V	V	V	V
Prosys OPC UA	Java	V	CVE-2022-30551	V	V	V
OPC UA Legacy Java Stack	Java	V	CVE-2022-30551	V	V	V
Kepware KEPServerEX	C/C++	V	V	V	V	CVE-2022-2848 CVE-2022-2825

How Did We Do That?

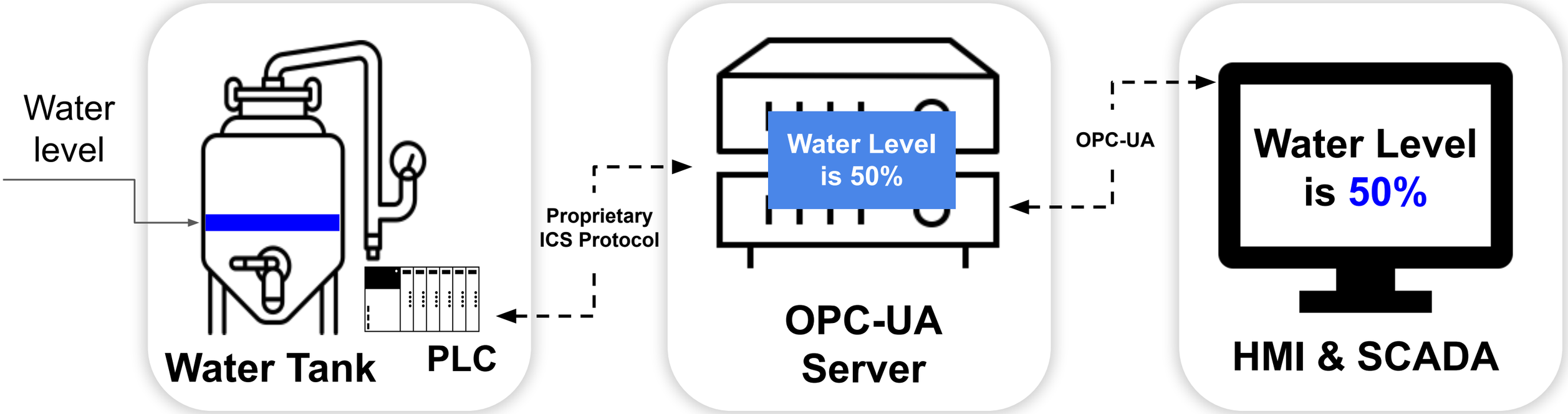
Agenda

- **What is OPC-UA?**
- Protocol Stack Implementations
- Bits and Bytes
- Research Methodology
- Vulnerabilities and Exploits
- OPC-UA Exploitation Framework
- Summary

What's the Problem?



What's the Problem?



What is OPC-UA?

Open Platform Communications - Unified Architecture

Protocol for data exchange between industrial devices and systems

- Server: stores tags/variables
- Client: requests tags/variables

Widely accepted standard for industrial communications

- Supported in Azure/AWS IoT cloud



OPC Foundation

OPC Foundation, specs first version ~2006

- opcfoundation.org

Lesson learned from “OPC Classic”

- Platform independent, scalable, secure

Detailed specifications

- **Information Model:** Object types, how to encode
- **Services:** Supported services such as read, write, etc
- **Security:** Authentication, authorization, encryption
- Many more

OPC 10000-1: UA Part 1: Overview and Concepts
OPC 10000-2: UA Part 2: Security
OPC 10000-3: UA Part 3: Address Space Model
OPC 10000-4: UA Part 4: Services
OPC 10000-5: UA Part 5: Information Model
OPC 10000-6: UA Part 6: Mappings
OPC 10000-7: UA Part 7: Profiles
OPC 10000-8: UA Part 8: DataAccess
OPC 10000-9: UA Part 9: Alarms and Conditions

Protocol Stacks and Frameworks

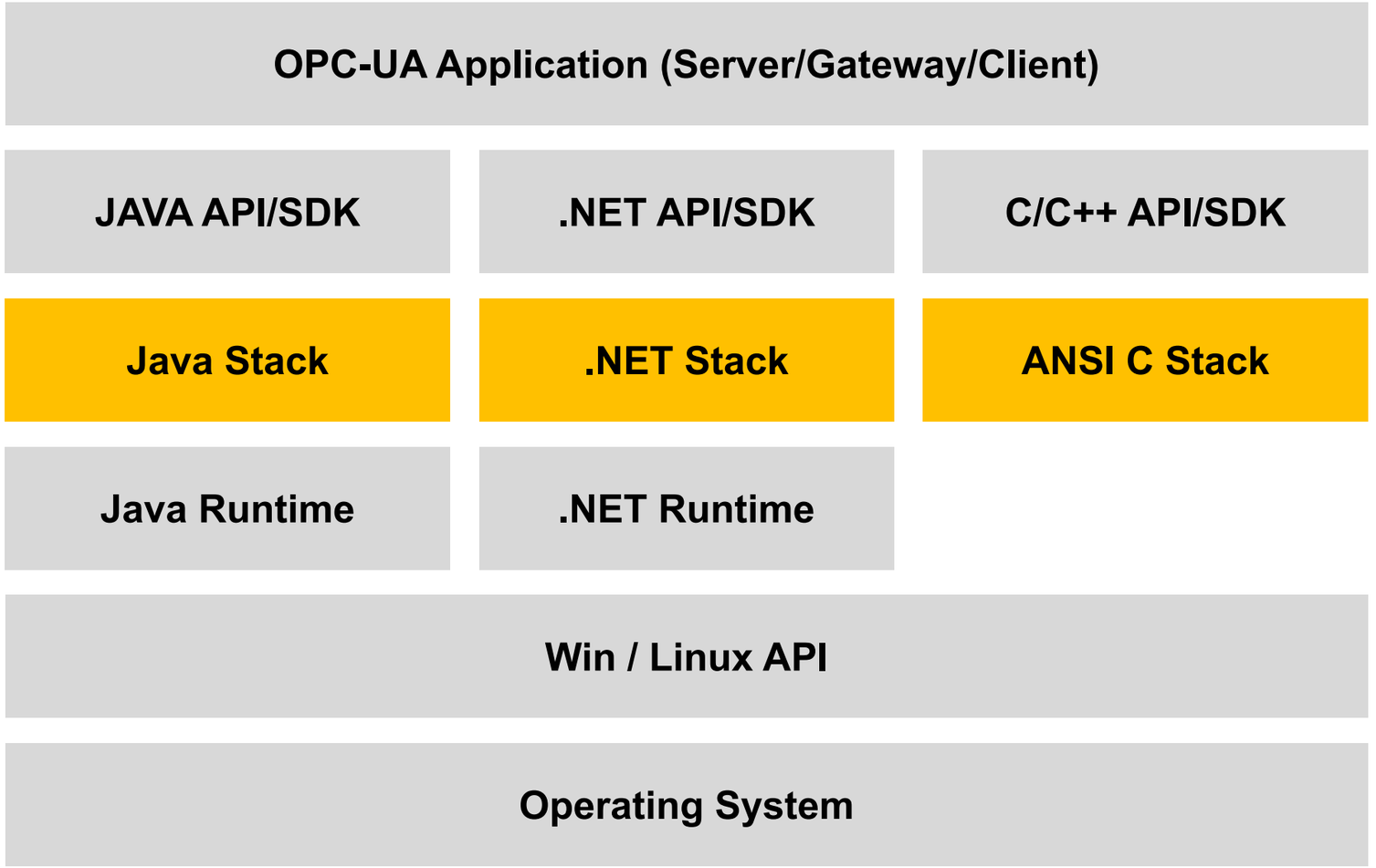
Agenda

- What is OPC-UA?
- **Protocol Stack Implementations**
- Bits and Bytes
- Research Methodology
- Vulnerabilities and Exploits
- OPC-UA Exploitation Framework
- Summary

OPC-UA Protocol Stacks

To expedite popularity, OPC Foundation created the first OPC-UA protocol stacks

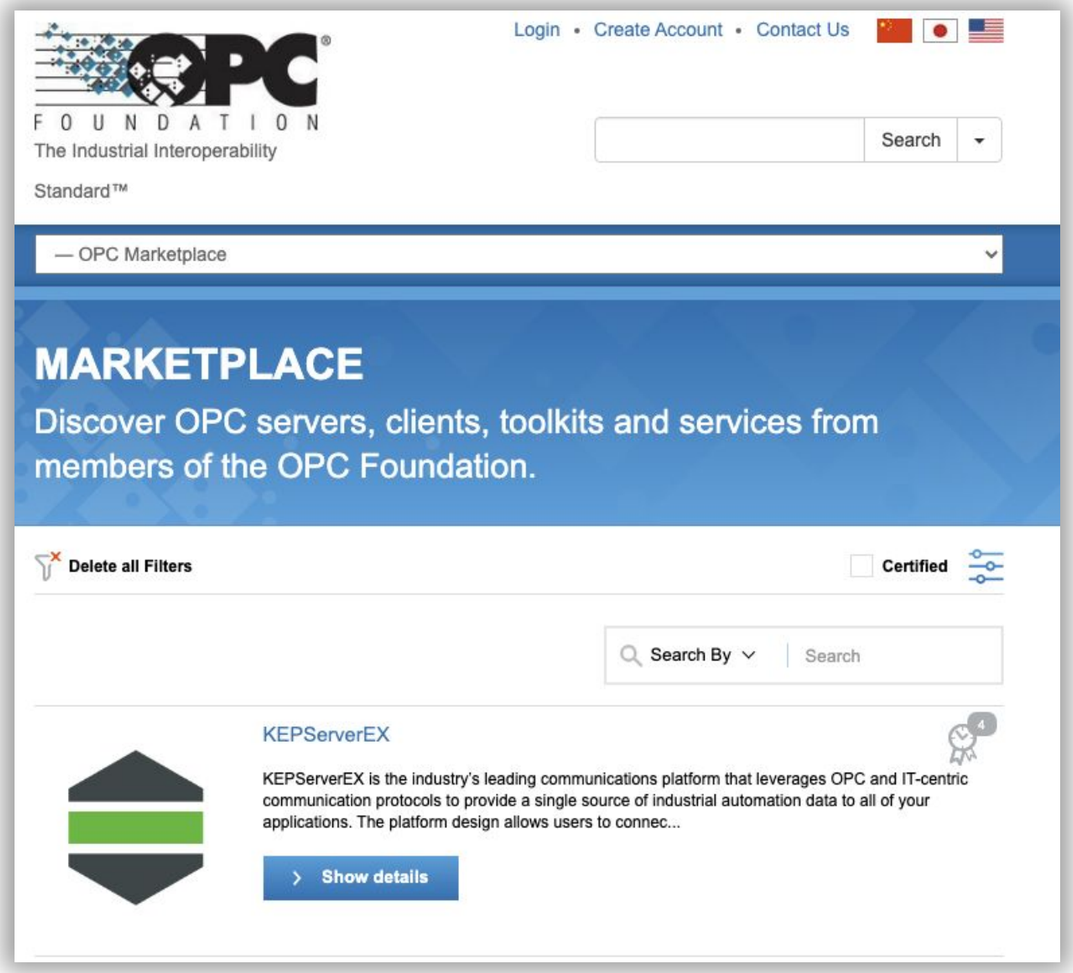
- ANSI C
- Java
- .NET



OPC-UA Supply Chain

With time, vendors integrated the base stacks and modified some of its code

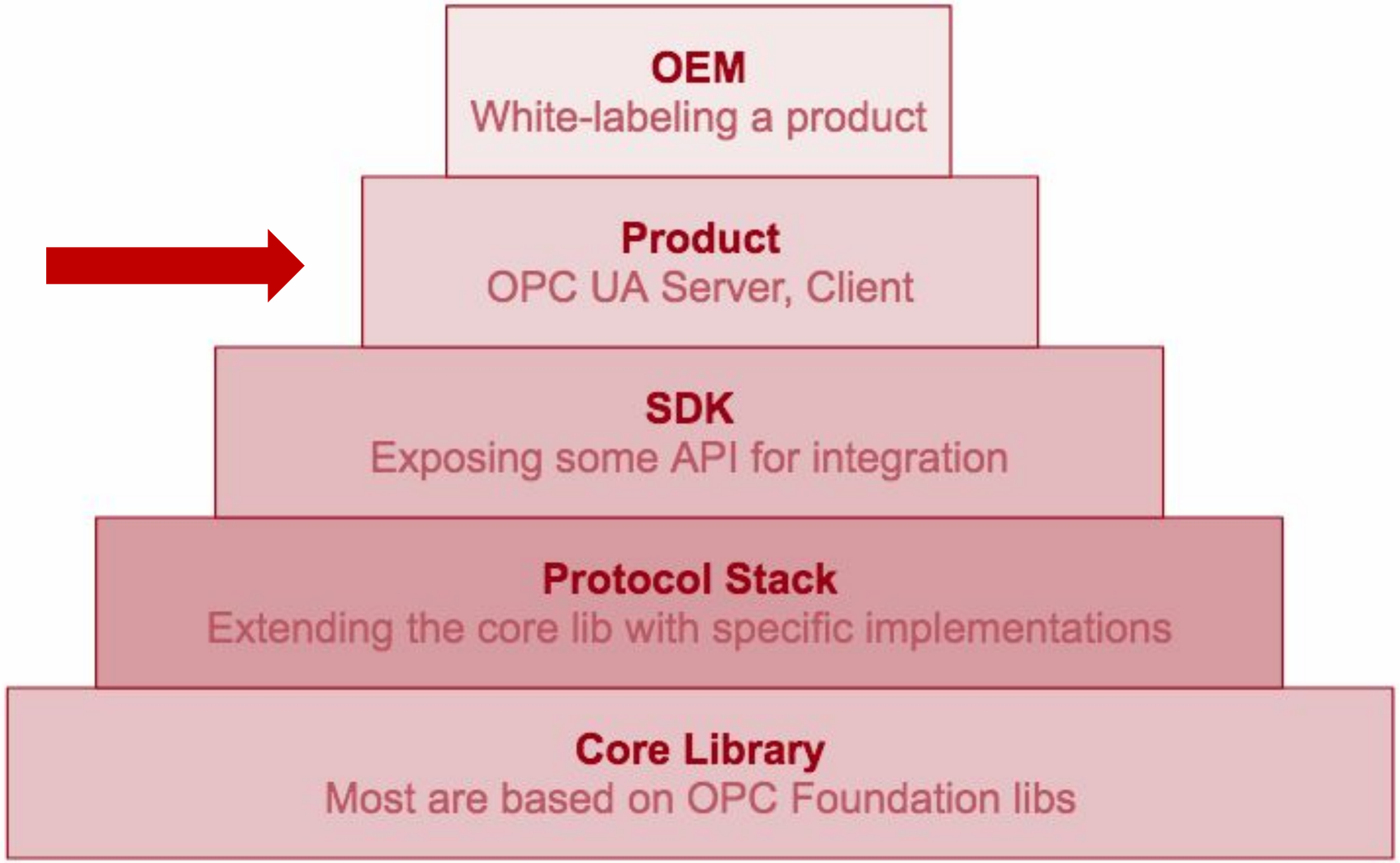
Currently, OPC Foundation lists more than 500 different products



https://opcfoundation.org/products

OPC-UA Supply Chain

The problem, is that most **products** are heavily relying on the base protocol stacks from OPC Foundation

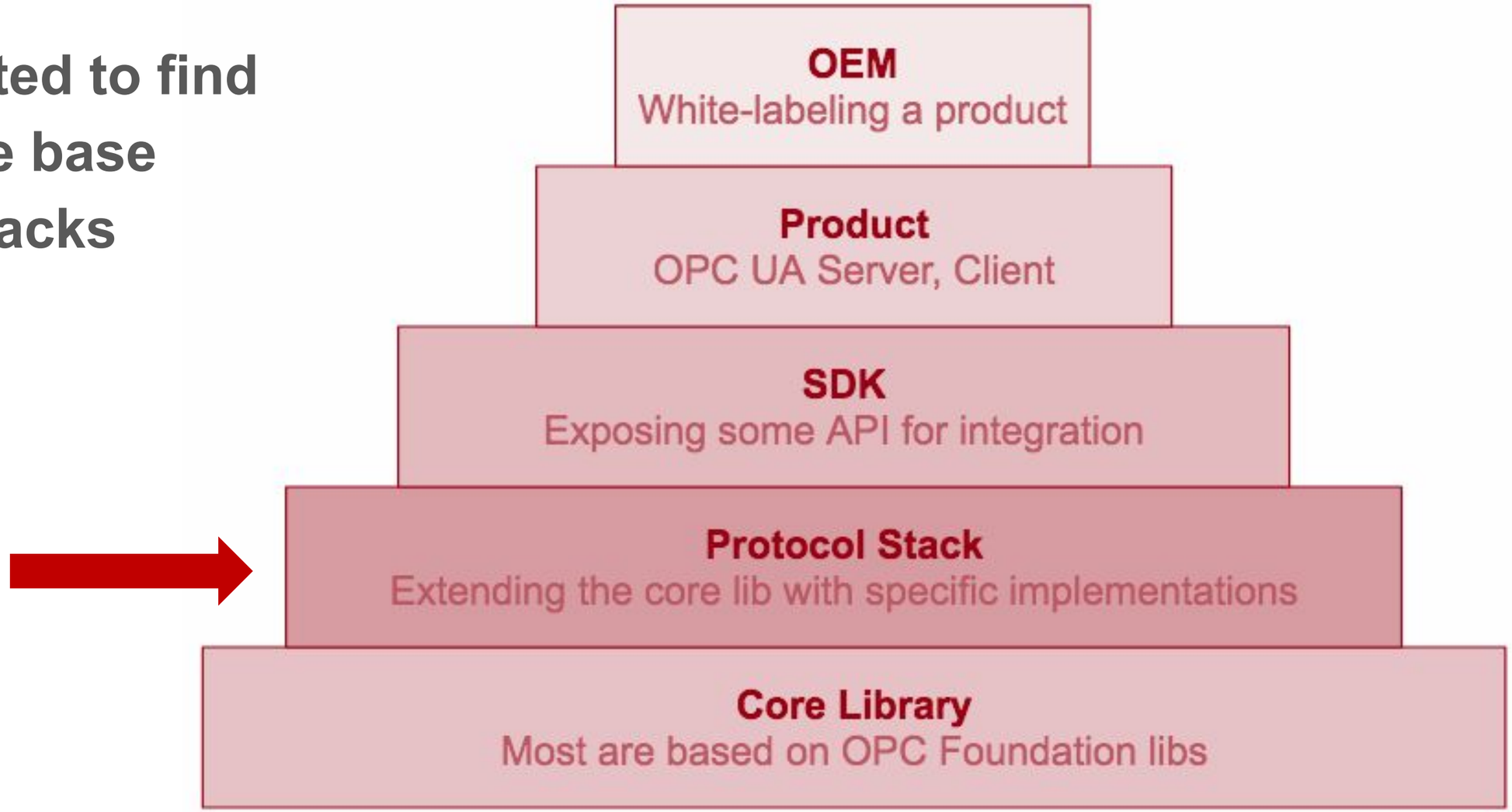


Top Products

	UA Automation C++ Server	OPC Foundation OPC UA .NET	Prosys OPC UA SDK for JAVA	Softing Integration Server	KEPServerEx
Extended Lib/SDK	Proprietary	OPC Foundation .NET	Proprietary	Proprietary	Proprietary
	+		+	+	
Core lib	OPC Foundation C++		OPC Foundation Java	OPC Foundation C	

Focus on the Protocol Stacks

So we wanted to find vulns in the base protocol stacks



Protocol Stacks

We also researched popular products such as:

- Softing Secure Integration Server
- PTC Kepware KEPServerEx
- Triangle Microworks SCADA Data Gateway
- Honeywell Matrikon
- Inductive Automation Ignition

OPC-UA Protocol Stack	Programing language	Is Open Source?
node-opcua	NodeJS	Yes
open62541	C	Yes
freeopcua (c++)	C++	Yes
python-opcua	Python	Yes
opcua-asyncio	Python	Yes
eclipse-milo	Java	Yes
ASNeG OpcUaStack	C++	Yes
locka99	Rust	Yes
Unified Automation	C++	No
OPC Foundation .NET Stack	C#	Yes
Softing OPC UA SDK	C++	No
Prosys OPC UA	Java	No
OPC UA Legacy Java Stack	Java	Yes
S2OPC	C	Yes
LibUA	C#	Yes

Bits and Bytes

Agenda

- What is OPC-UA?
- Protocol Stack Implementations
- **Bits and Bytes**
- Research Methodology
- Vulnerabilities and Exploits
- OPC-UA Exploitation Framework
- Summary

OPC-UA Nodes

Everything is a node

- Variable (e.g. “Water Level”)
- Type of the Variable value (e.g. Float)

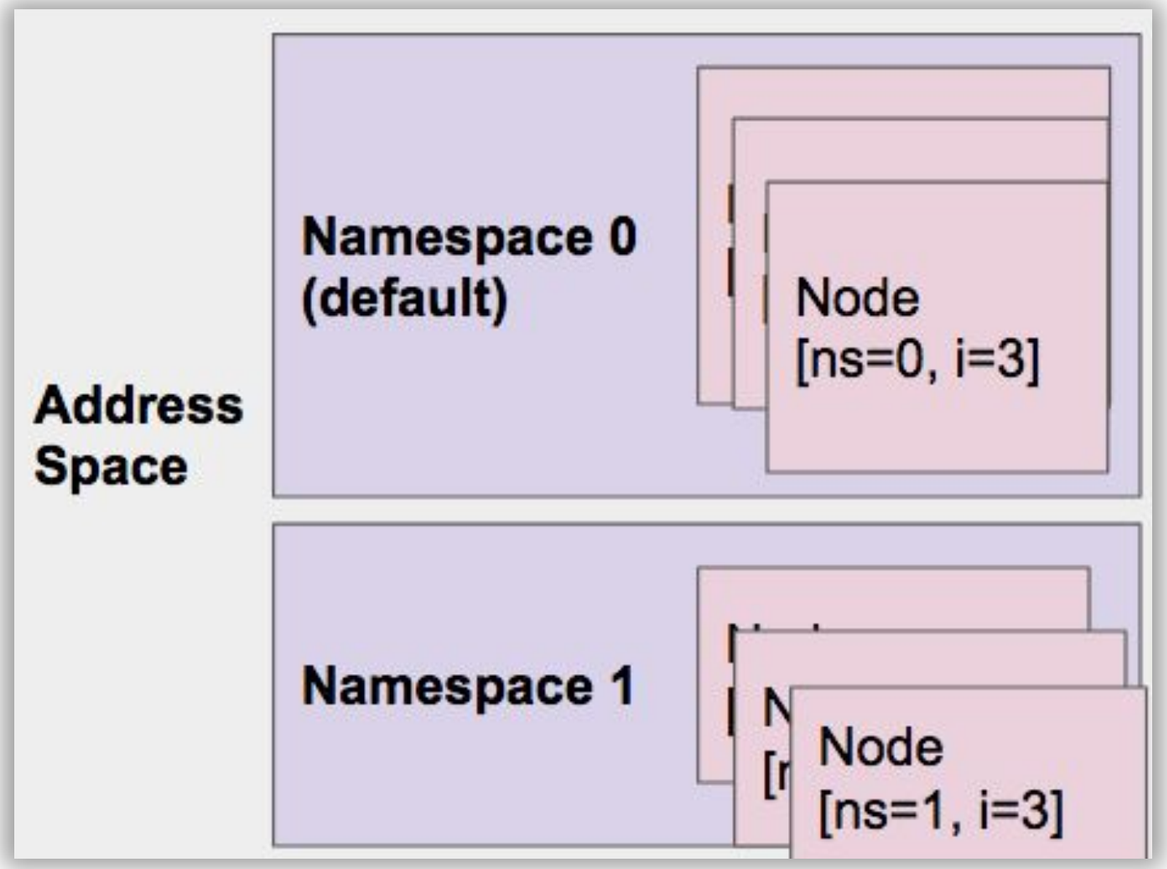
Nodes are identified by [ns, i]

- NodeID (i=1)
- Namespace ID (ns=0)

Namespace is a container for nodes

- Namespace 0: default namespace and contains the default nodes

Address Space provide a standard way for servers to represent objects to clients



OPC-UA Services

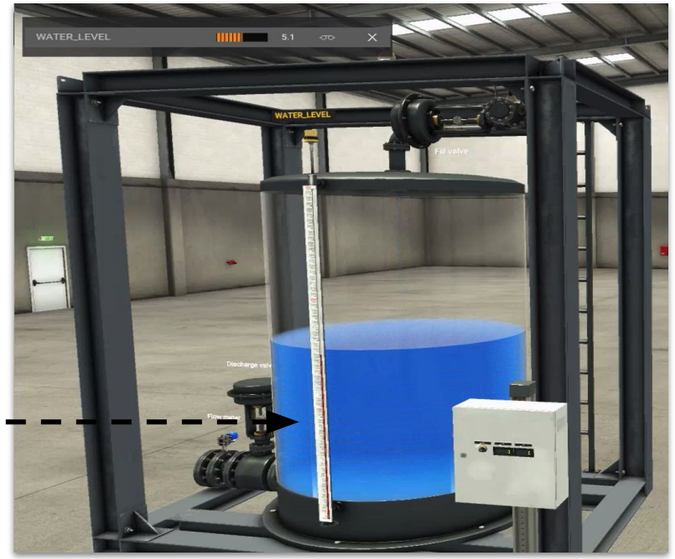
Our interaction with the server is via request/response fashion. In most cases we are doing some “action” on nodes.
Examples:

Service Set	Service Name	Description
Attribute	Read Service	Read values from attributes of nodes
	Write Service	Write values to attributes of nodes
Method	Call Service	Call (invoke) a list of methods.
View	Browse	Navigate through the AddressSpace - find Node references

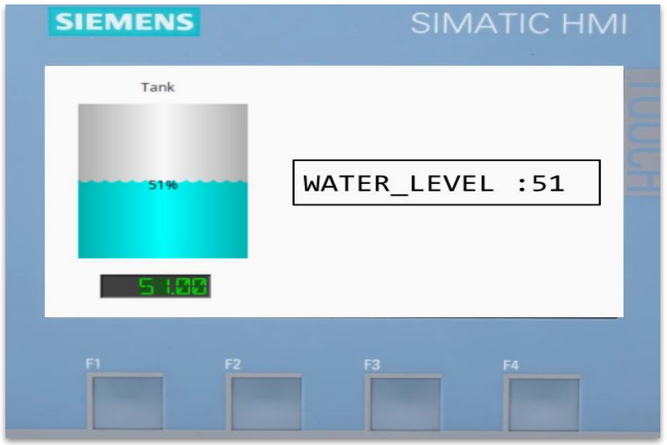
OPC 10000-4: UA Part 4: Services

Example

Node Name	Node Class and Type
Fill Valve	**Variable** with **DataType** Boolean
Discharge Valve	**Variable** with **DataType** Boolean
Flow Meter	**Variable** with **DataType** Float
Water Level	**Variable** with **DataType** Float
Start/Stop	Method



Tank (water level %50)

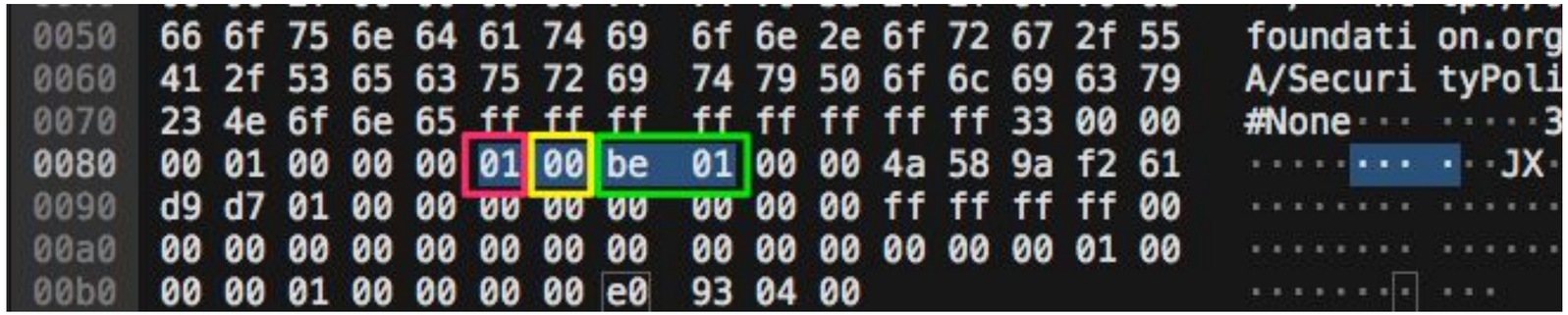


Nodes Encoding [ns=0, i=446]

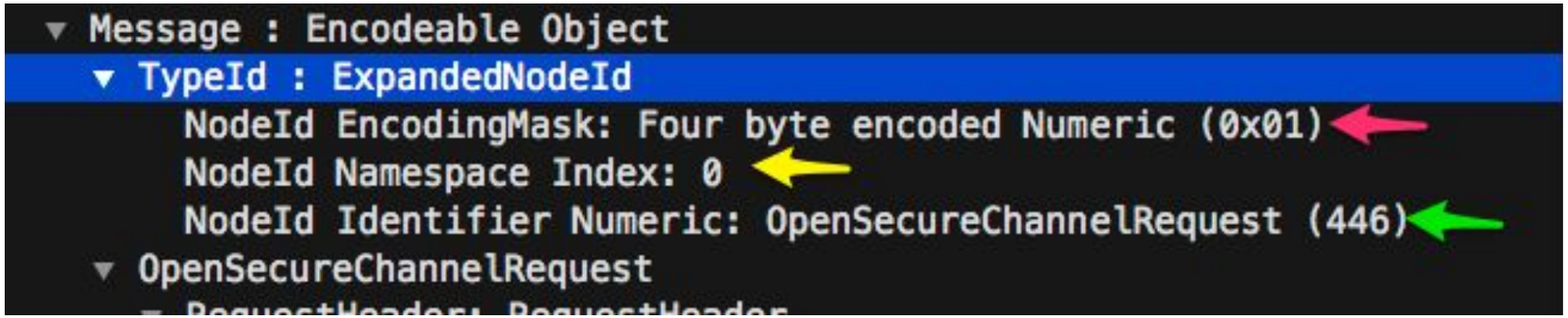
Table 9 – Four Byte NodeId Binary DataEncoding

Name	Data Type	Description
Namespace	Byte	The *Namespace* shall be in the range 0 to 255.
Identifier	UInt16	The *Identifier* Type is 'Numeric'. The *Identifier* shall be an integer in the range 0 to 65 535.

Specifications

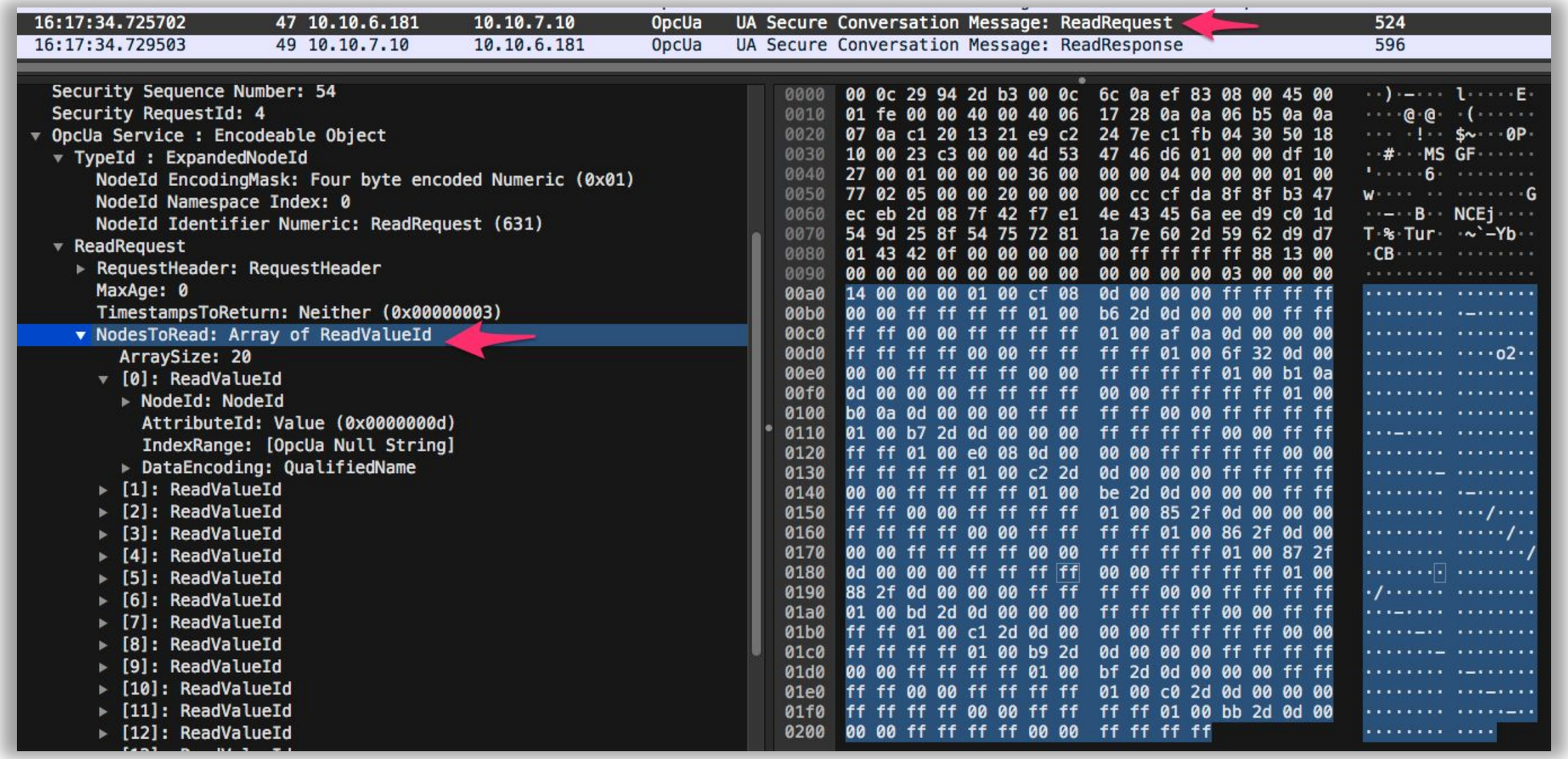


Binary Representation



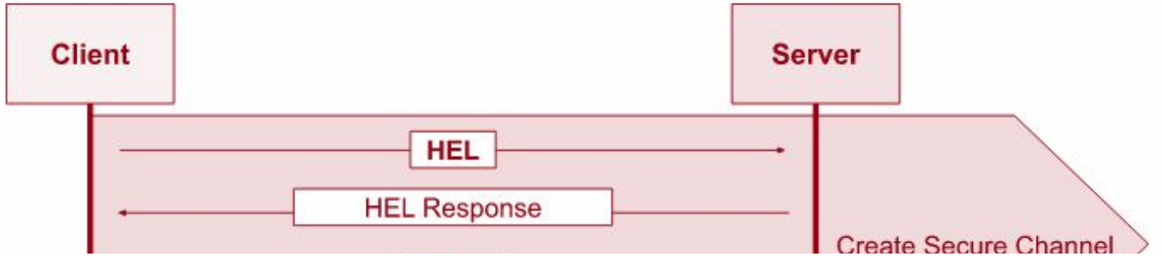
Binary Parsing

Example: Read Service: Reading 12 Nodes



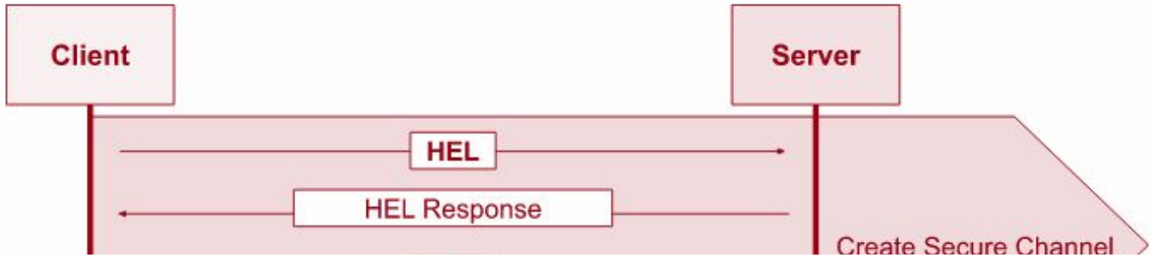
HEL

HEL: Hello message



HEL

HEL: Hello message



Endpoint URL

- Scheme - must be opc.tcp or opc.https
- Server address
- Port
- Discovery endpoint

opc.tcp://SERVER_IP:62541/UA/Server

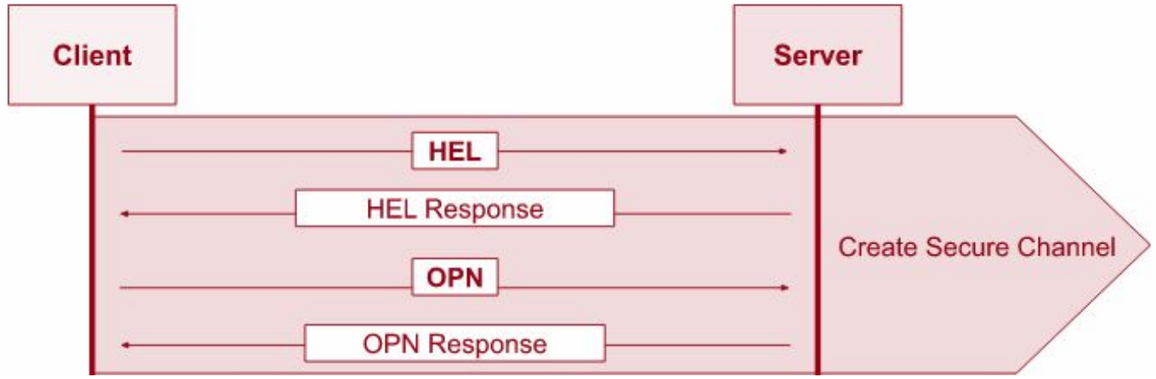
```
Frame 1: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits) on interfac
Ethernet II, Src: Eve_0a:ef:83 (00:0c:6c:0a:ef:83), Dst: VMware_0d:ed:50 (00:0c:29
Internet Protocol Version 4, Src: 10.10.6.181, Dst: 10.10.7.11
Transmission Control Protocol, Src Port: 49422, Dst Port: 62541, Seq: 1, Ack: 1, L
OpcUa Binary Protocol
    Message Type: HEL
    Chunk Type: F
    Message Size: 91
    Version: 0
    ReceiveBufferSize: 65536
    SendBufferSize: 65536
    MaxMessageSize: 16777216
    MaxChunkCount: 5000
    EndpointUrl: opc.tcp://desktop-eq75855:62541/Quickstarts/ReferenceServer
```

```
0000  00 0c 29 0d ed 50 00 0c  6c 0a ef 83 08 00 45 00   ··)··P·· l·····E·
0010  00 83 00 00 40 00 40 06  18 a2 0a 0a 06 b5 0a 0a   ····@·@· ········
0020  07 0b c1 0e f4 4d 9e dc  7b 99 2b d3 5e 98 50 18   ·····M·· {·+·^·P·
0030  10 00 22 49 00 00 48 45  4c 46 5b 00 00 00 00 00   ··"I··HE LF[·····
0040  00 00 00 00 01 00 00 00  01 00 00 00 00 01 88 ··   ········ ········
0050  00 00 3b 00 00 00 6f 70  63 2e 74 63 70 3a 2f 2f   ··;···op c.tcp://
0060  64 65 73 6b 74 6f 70 2d  65 71 37 35 38 35 35 3a   desktop- eq75855:
0070  36 32 35 34 31 2f 51 75  69 63 6b 73 74 61 72 74   62541/Qu ickstart
0080  73 2f 52 65 66 65 72 65  6e 63 65 53 65 72 76 65   s/Refere nceServe
0090  72                                                r
```

OPN

HEL: Hello message

OPN: OpenSecureChannel message



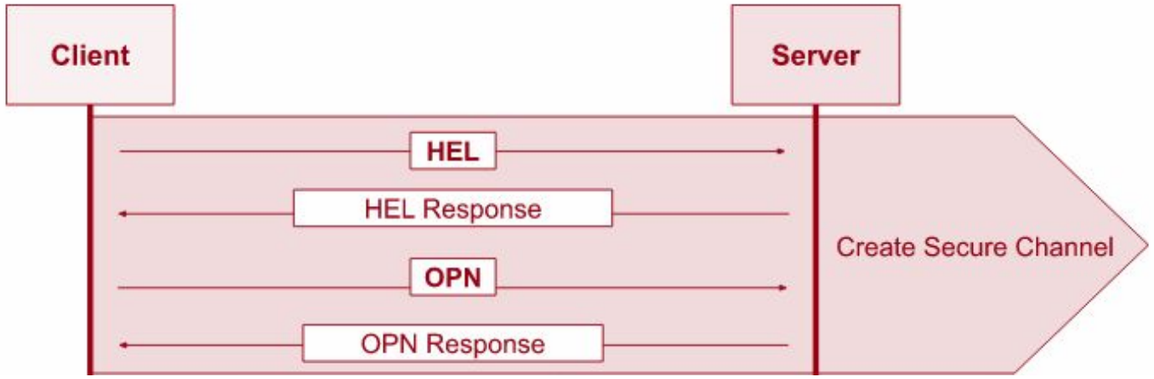
OPN

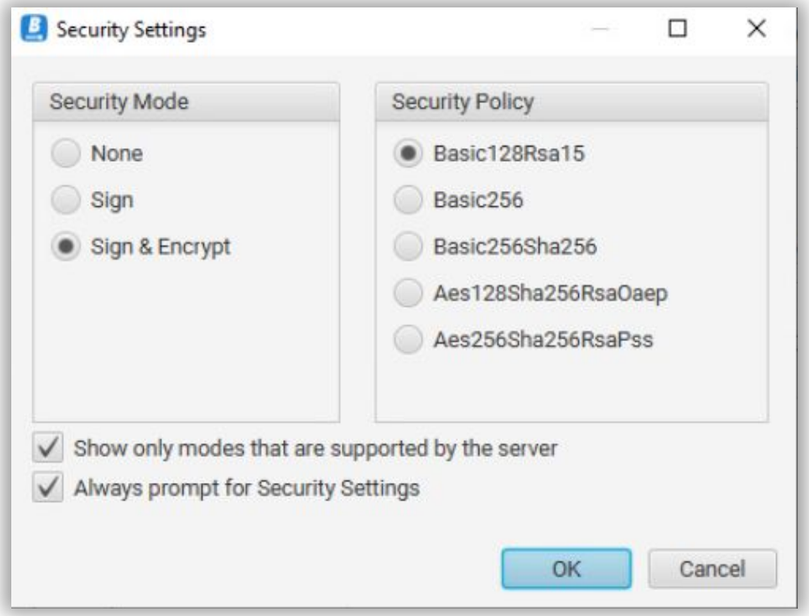
HEL: Hello message

OPN: OpenSecureChannel message

Security Mode

- None
- Sign
- Sign & Encrypt





SecurityPolicies supported by Prosys OPC-UA server

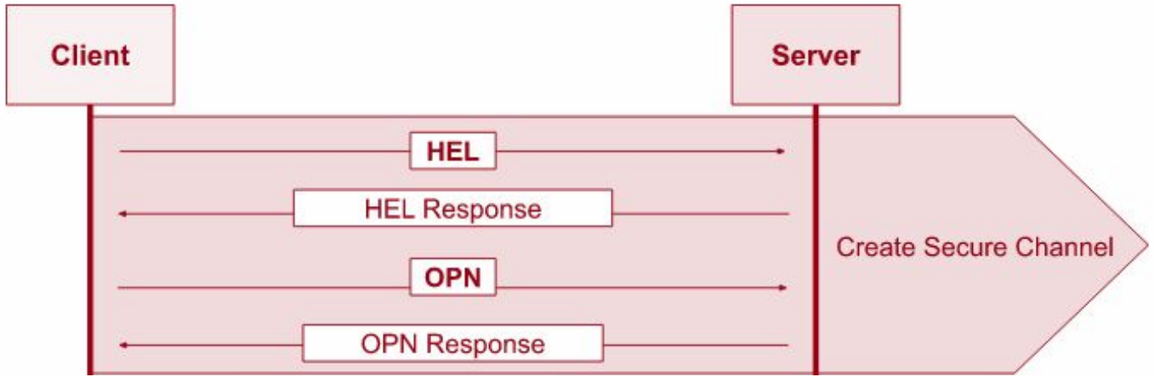
OPN

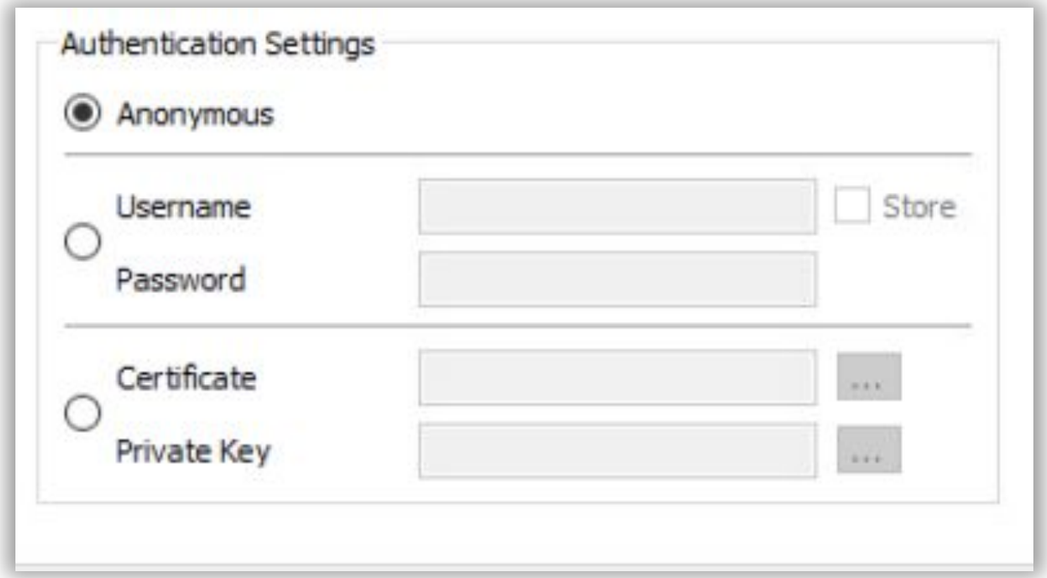
HEL: Hello message

OPN: OpenSecureChannel message

Authentication

- Anonymous
- Username/password
- Certificate





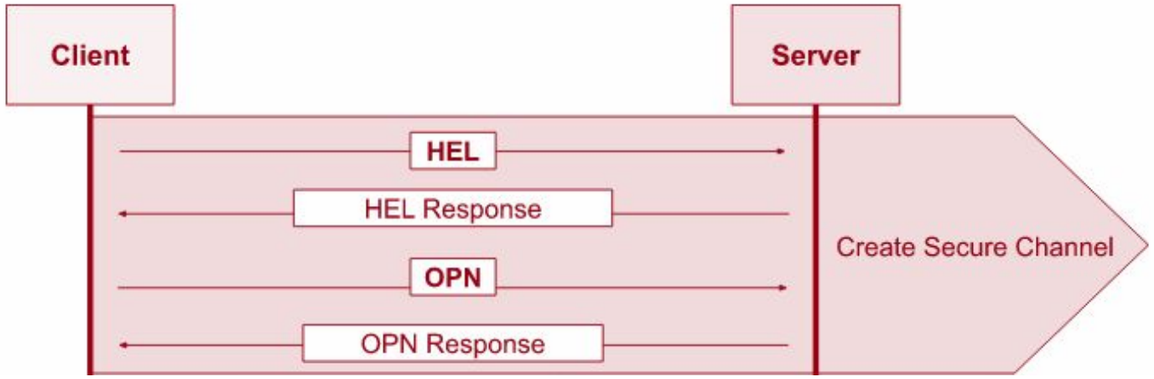
Authentication settings for an OPC-UA client, shown using UAExeprt

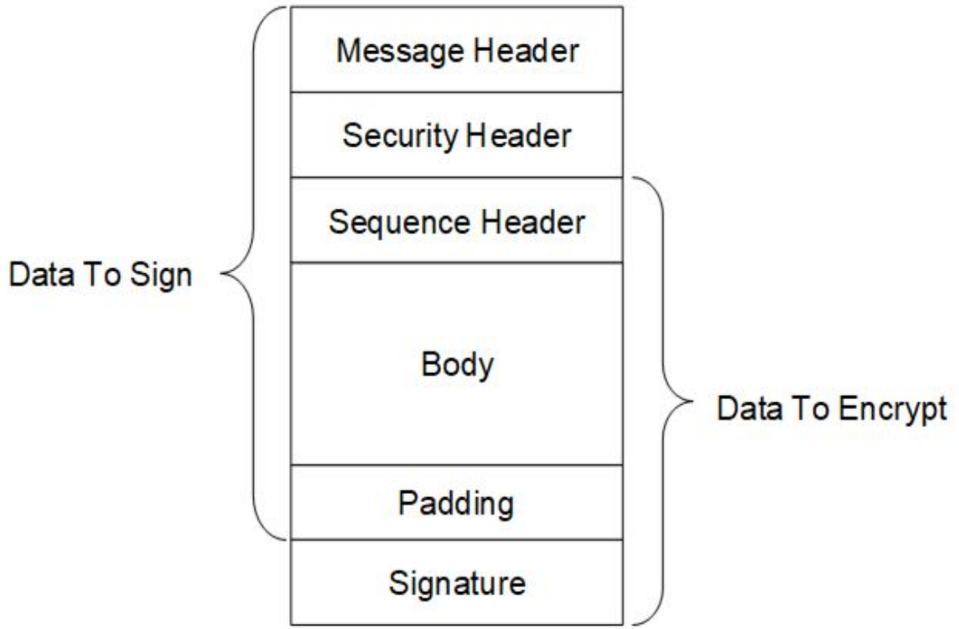
OPN

HEL: Hello message

OPN: OpenSecureChannel message

- Security Mode and Policy
- Authentication





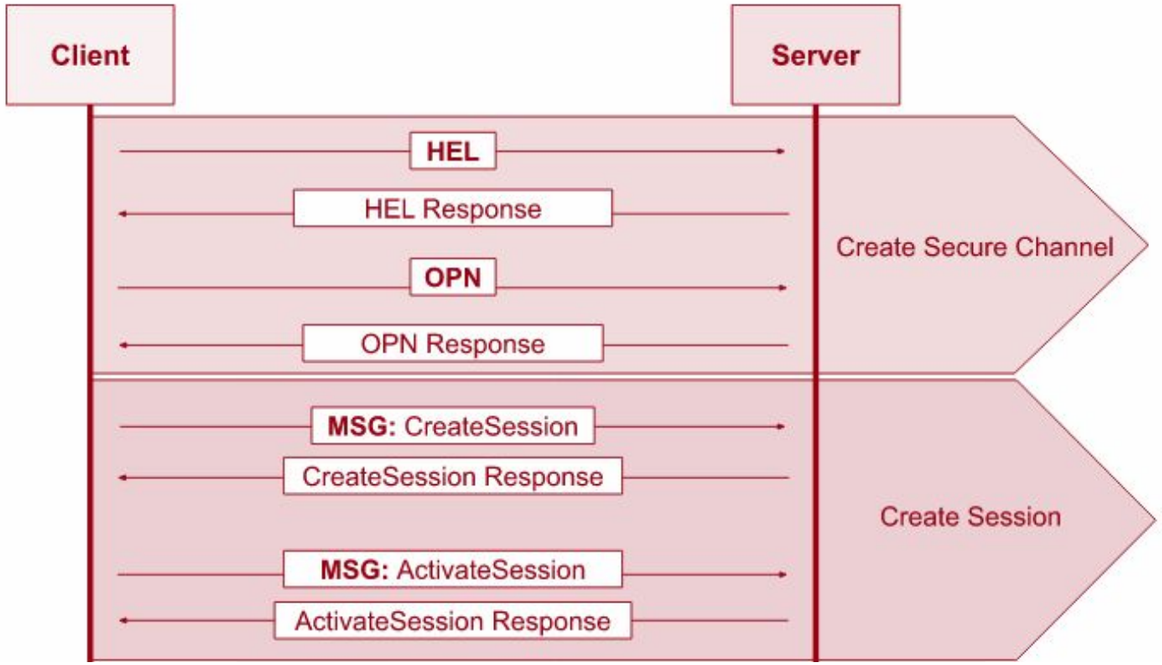
OPC UA Secure Conversation MessageChunk

CreateSession

HEL: Hello message

OPN: OpenSecureChannel message

MSG: A generic message container. Some service will be used.



CreateSession

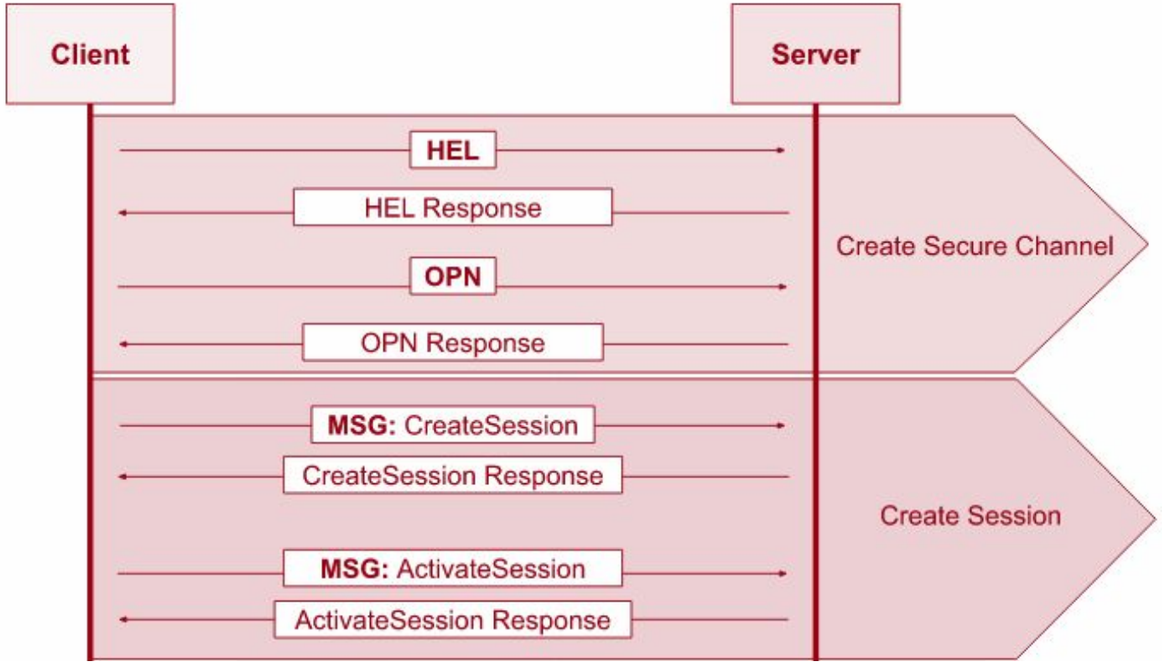
HEL: Hello message

OPN: OpenSecureChannel message

MSG: A generic message container. Some service will be used.

Create Session + Activate

- Configure the session (e.g. timeout, message size, etc)




```
CreateSessionRequest
  > RequestHeader: RequestHeader
  > ClientDescription: ApplicationDescription
    ServerUri: [OpcUa Null String]
    EndpointUrl: opc.tcp://desktop-ad29i88:62541/Quickstar
    SessionName: [OpcUa Empty String]
    ClientNonce: 0000000000000000000000000000000000000000
    ClientCertificate: <MISSING>[OpcUa Null ByteString]
    RequestedSessionTimeout: 1200000
    MaxResponseMessageSize: 16777216
```

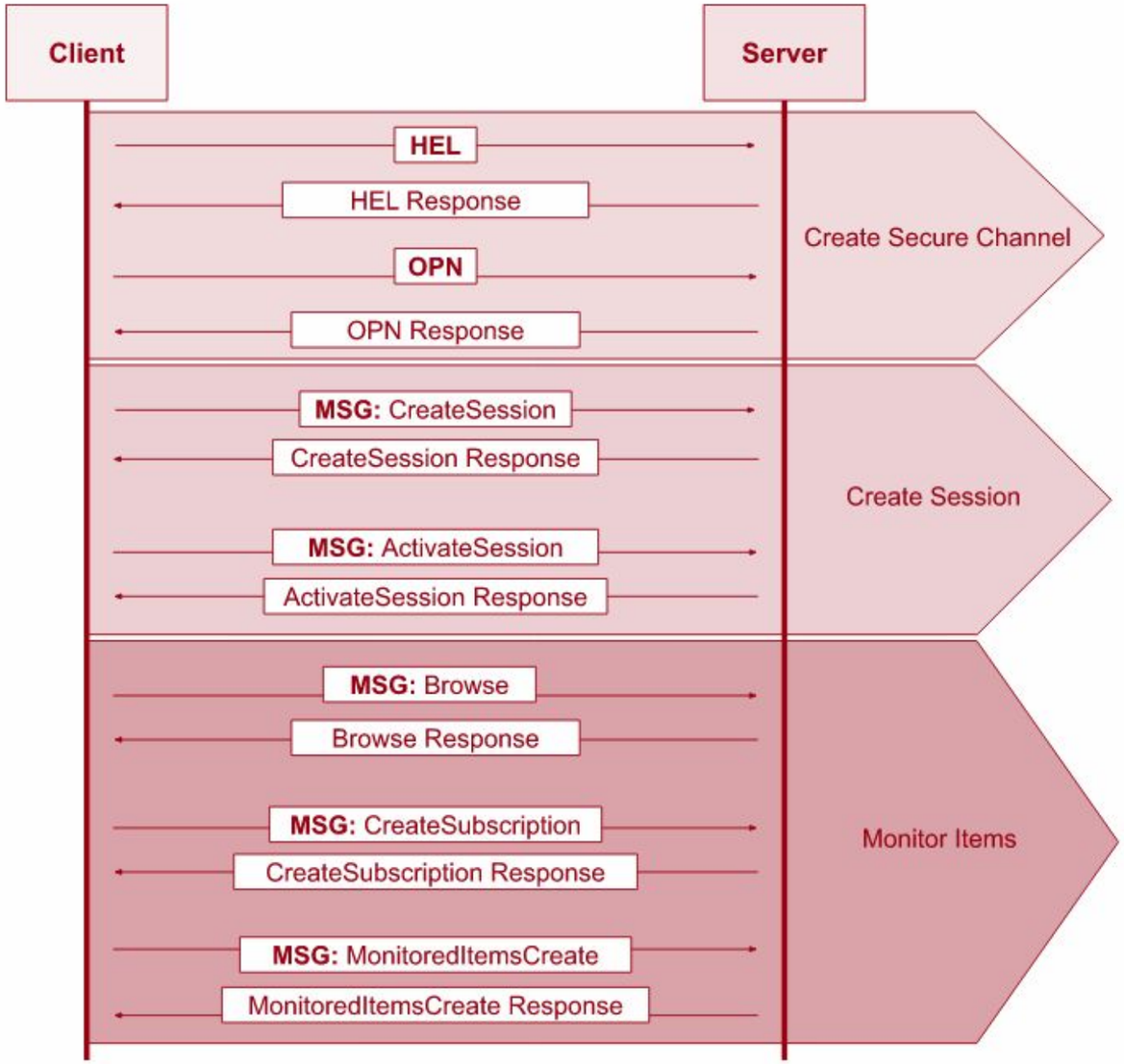
Full Session

HEL: Hello message

OPN: OpenSecureChannel message

MSG: A generic message container (secured with the channel's keys)

CLO: CloseSecureChannel message



Research Methodology

Agenda

- What is OPC-UA?
- Protocol Stack Implementations
- Bits and Bytes
- **Research Methodology**
- Vulnerabilities and Exploits
- OPC-UA Exploitation Framework
- Summary

Building Basic OPC-UA Client

Why?

- Hands-on
- Focus on logic
- Customizable to our vuln research needs

How?

- Specification
- Protocol analysis + Wireshark

```
opcua = OPCUA(ip_addr, port, query_string)
opcua.session_timeout = 3600 * 1000 # 1hr
opcua.requested_lifetime = 3600 * 1000 # 1hr
opcua.max_chunk_size = max_chunk_size
opcua.create_session()
```

- FreeOpcUa Python OPC-UA (Python)
- Prosys OPC-UA Browser (Java)
- Unified Automation UaExpert (C/C++)

Building the Setup

Intel NUC x 2

- Intel Core i7-1165G7
- 32 GB RAM

Installed VMware ESXi

Prepared a Windows 10 x64 Image
~10 machines per NUC





Installing & Configuring Targets

Protocol Stack Libraries

- Unified Automation - ANSI C Stack - C
- OPC Foundation - .NET Standard - .NET
- OPC Foundation - Java Legacy - Java
- Prosys OPC UA SDK for Java - Java
- FreeOpcUA opcua-asyncio Python
- Eclipse Milo - Java
- Node-opcua - Node JS
- Open62541 - C
- OPC UA rust – Rust

OPC UA Servers

- Inductive Automation Ignition
- Unified Automation UaGateway
- PTC Kepware KepServerEx
- Prosys OPC UA Simulation Server
- Softing edgeConnector

Gateways

Triangle Microworks SCADA Data Gateway

Softing Secure Integration Server

Clients

PTC Kepware KepServerEx

Prosys OPC UA Browser

Softing edgeAggregator

Inductive Automation Ignition











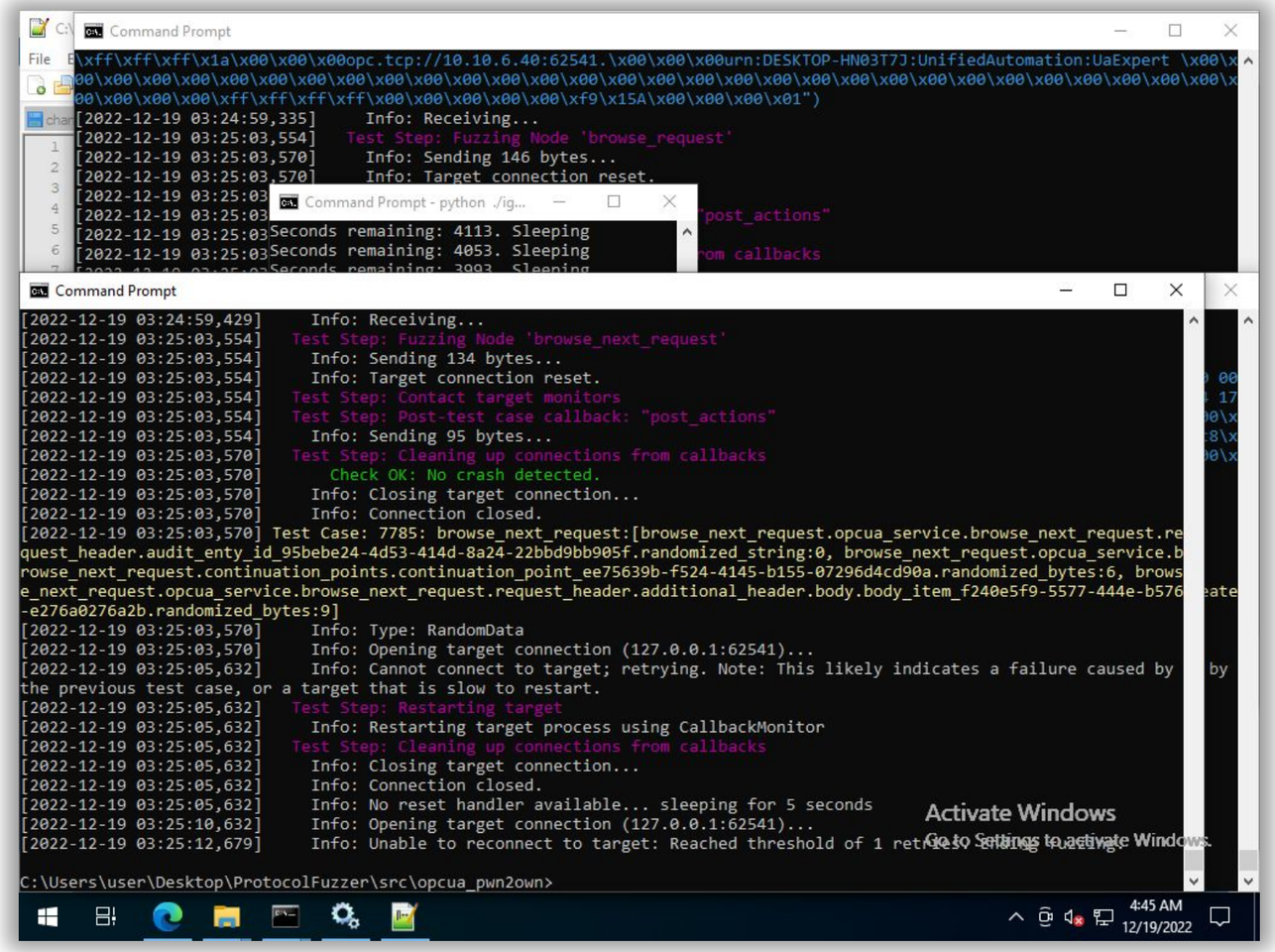
Network Fuzzer

Released open-source OPC-UA fuzzer, based on boofuzz

Found 2 heap/stack overflow

Fuzzing 6 Services

- Read Service
- Browse Service
- Browse Next Service
- Create Subscription Service
- Add Nodes Service
- History Read Service



https://github.com/claroty/opcua_network_fuzzer

Fuzzers: Coverage Based

Found old source-code for ANSI C OPC-UA stack

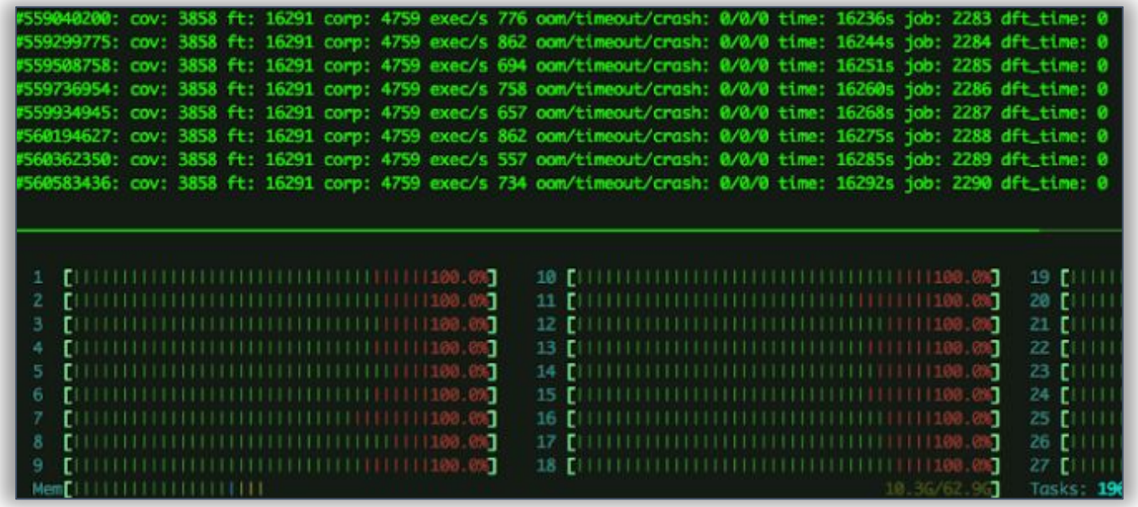
Used both libFuzzer / AFL

Wrote small harness, mostly to fuzz the decode routines

https://github.com/linshenqi/UA-AnsiC

```
1747    #ifdef __AFL_HAVE_MANUAL_CONTROL
1748    __AFL_INIT();
1749    #endif
1750    unsigned char *pData = __AFL_FUZZ_TESTCASE_BUF;
1751    while (__AFL_LOOP(100000)) {
1752        // Size = __AFL_FUZZ_TESTCASE_LEN;
1753        int Size = __AFL_FUZZ_TESTCASE_LEN;
1754        OpcUa_MessageContext_Initialize(&cContext);
1755
1756        cContext.KnownTypes = &OpcUa_ProxyStub_g_EncodeableTypes;
1757        cContext.NamespaceUris = &OpcUa_ProxyStub_g_NamespaceUris;
1758        cContext.AlwaysCheckLengths = OpcUa_False;
1759
1760        uStatus = OpcUa_MemoryStream_CreateReadable(pData, Size, &p
1761        if (uStatus != OpcUa_Good) continue;
1762        uStatus = OpcUa_BinaryDecoder_Create(&pDecoder);
```

AFL harness



libFuzzer burning CPUs

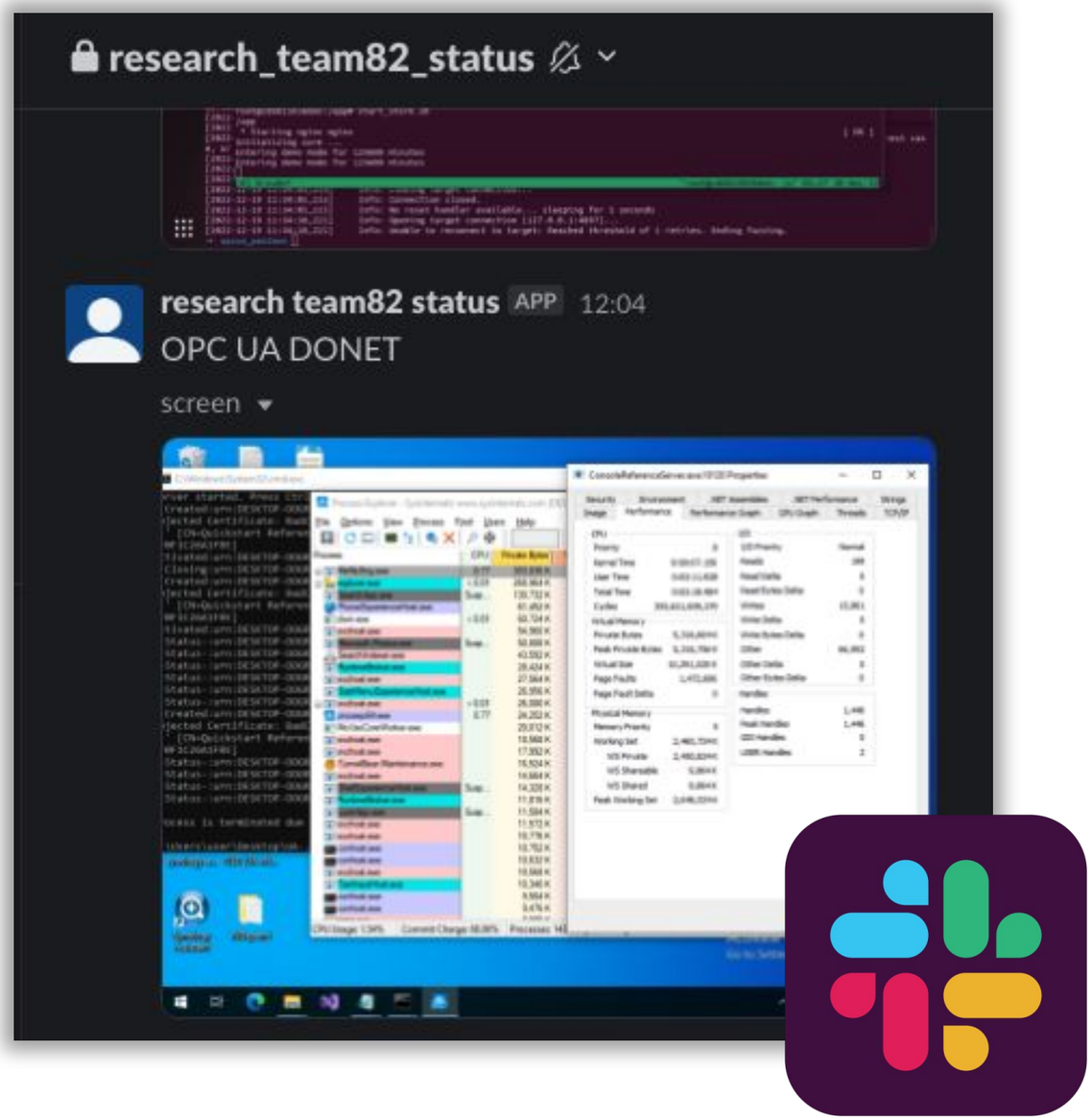
Control the Fuzzers

Dozens of fuzzers running

- **Network** based: using boofuzz
- **Memory/Coverage** based: using AFL, libfuzzer
- **Closed binary:** using WinAFL, UnicornAFL (CPU Emulator)

Monitored everything through Slackbot

Collected millions of corpus



Specs & RE

Looking for esoteric and complex features/mechanisms

What will developers overlook?

Reverse engineer and code review to observe different implementations

Pre-auth (HEL, OPN) vs post-auth

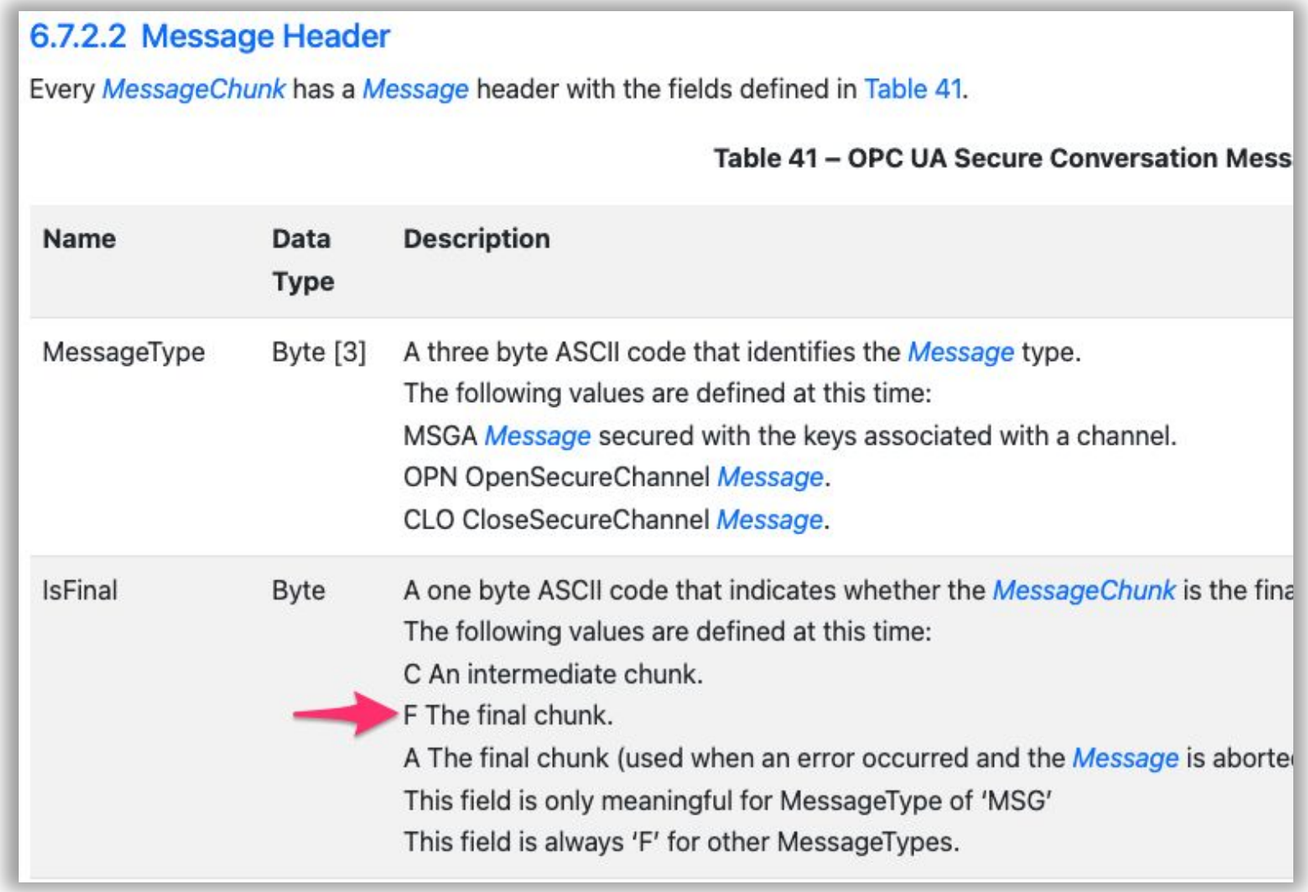
Specs & RE

Looking for esoteric and complex features/mechanisms

What will developers overlook?

Reverse engineer and code review to observe different implementations

Pre-auth (HEL, OPN) vs post-auth



6.7.2.2 Message Header

Every *MessageChunk* has a *Message* header with the fields defined in Table 41.

Table 41 – OPC UA Secure Conversation Mess

Name	Data Type	Description
MessageType	Byte [3]	A three byte ASCII code that identifies the *Message* type. The following values are defined at this time: MSGA *Message* secured with the keys associated with a channel. OPN OpenSecureChannel *Message*. CLO CloseSecureChannel *Message*.
IsFinal	Byte	A one byte ASCII code that indicates whether the *MessageChunk* is the fina The following values are defined at this time: C An intermediate chunk. F The final chunk. A The final chunk (used when an error occurred and the *Message* is aborte This field is only meaningful for MessageType of 'MSG' This field is always 'F' for other MessageTypes.

What happens if we are not sending the Final flag?

https://reference.opcfoundation.org/v104/Core/docs/Part6/6.7.2/

Specs & RE

Looking for esoteric and complex features/mechanisms

What will developers overlook?

Reverse engineer and code review to observe different implementations

Pre-auth (HEL, OPN) vs post-auth

Table 19 – CloseSession Service Parameters

Name	Type	Description
Request		
requestHeader	RequestHeader	Common request parameters (see 7.28 for *RequestHeader* definition).
deleteSubscriptions	Boolean	If the value is TRUE, the Server deletes all Subscriptions associated with the Session. If the value is FALSE, the Server keeps the Subscriptions associated with the Session until they timeout based on their own lifetime.

What happens if we keep all subscriptions alive?

https://reference.opcfoundation.org/Core/Part4/v104/docs/5.6.4

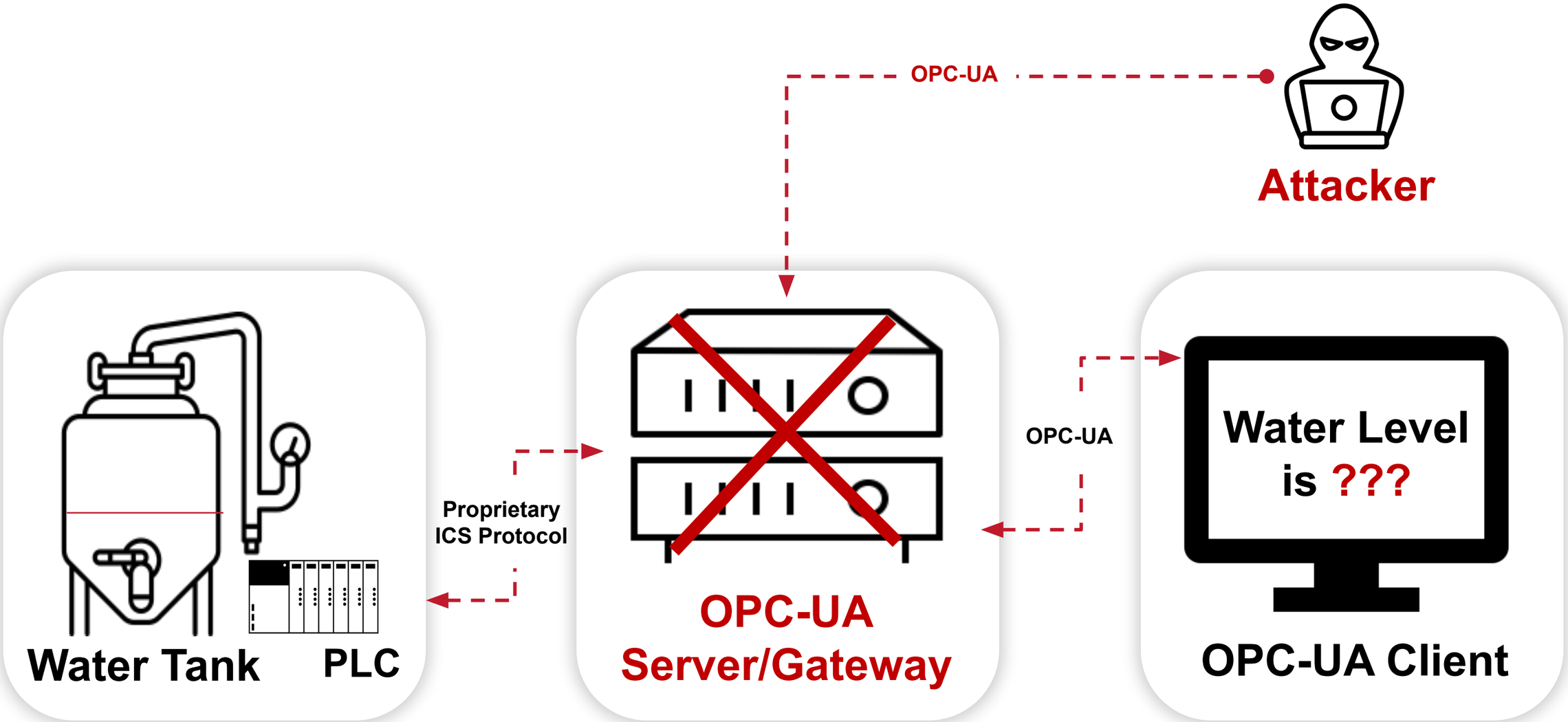
Vulnerabilities and Exploits

Denial of Service - Servers

Agenda

- What is OPC-UA?
- Protocol Stack Implementations
- Bits and Bytes
- Research Methodology
- **Vulnerabilities and Exploits**
- OPC-UA Exploitation Framework
- Summary

OPC-UA Server - Denial of Service



Denial of Service - Vectors

- Resource exhaustion: uncontrolled memory management
- Threads deadlock
- Use after free bugs
- Buffer overflows: heap/stack corruption
- Uncaught exceptions
- Busy loops / unlimited recursions: call-stack overflow

Denial of Service – Attack Concepts

Resource exhaustion - uncontrolled memory management

- Chunk Flooding
- Unlimited ConditionRefresh Attack
- Unlimited Persistent Monitored Subscriptions
- Unlimited Open Channels

Threads deadlock

- Worker Starvation

Use-after-free bugs

- Method Calling From Dead Session
- Add/Remove From Namespace While Browsing

Buffer overflows - heap/stack corruption

- Unicode Conversion - OOB Write

Uncaught exceptions

- Parser Bug - Dissecting Malformed OPC-UA Data Type

Busy loops / unlimited recursions – call-stack overflow

- Complex Deep Nested Variants (OTORIO)
- Certificate Chain Loop (Sector7)
- Unlimited Translate Browse Path (JFrog)

Denial of Service – Attack Concepts

Resource exhaustion - uncontrolled memory management

- **Chunk Flooding**
- Unlimited ConditionRefresh Attack
- Unlimited Persistent Monitored Subscriptions
- Unlimited Open Channels

Threads deadlock

- Worker Starvation

Use-after-free bugs

- Method Calling From Dead Session
- Add/Remove From Namespace While Browsing

Buffer overflows - heap/stack corruption

- Unicode Conversion - OOB Write

Uncaught exceptions

- Parser Bug - Dissecting Malformed OPC-UA Data Type

Busy loops / unlimited recursions – call-stack overflow

- Complex Deep Nested Variants (OTORIO)
- Certificate Chain Loop (Sector7)
- Unlimited Translate Browse Path (JFrog)

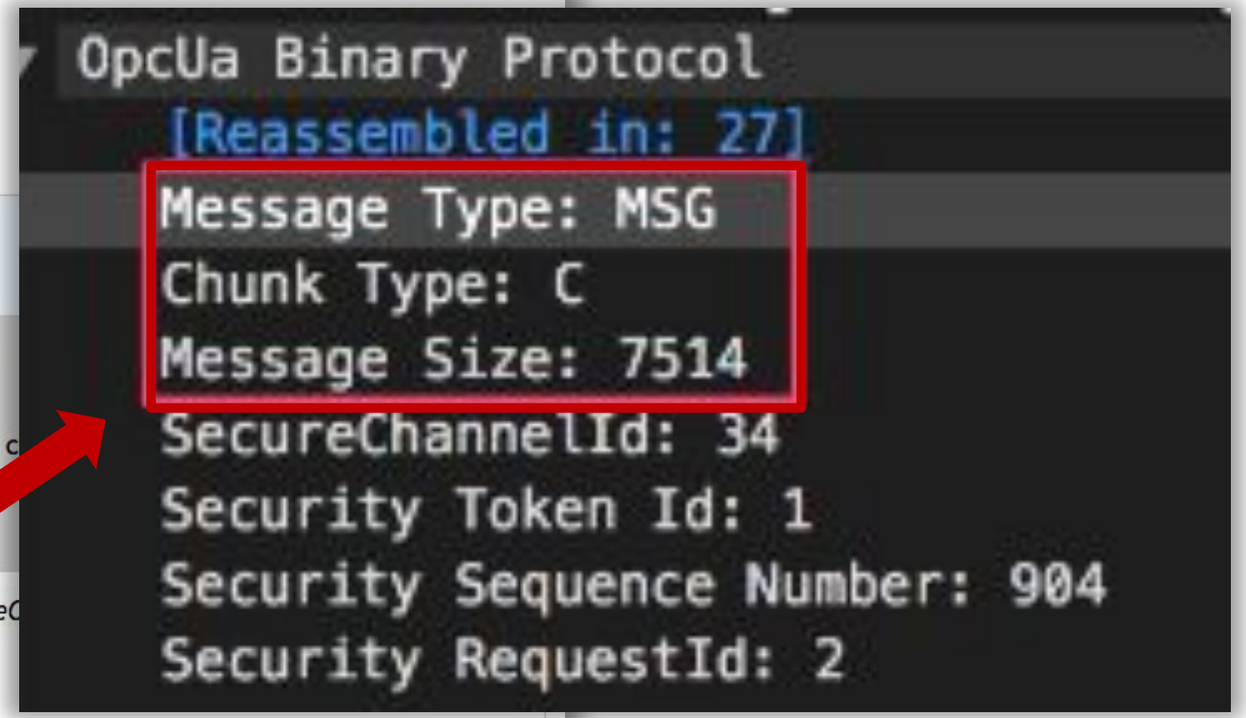
Denial of Service - Chunk Flooding

6.7.2.2 Message Header ↑

Every *MessageChunk* has a *Message* header with the fields defined in Table 41.

Table 41 – OPC UA Secure Conversation Message header

Name	Data Type	Description
MessageType	Byte [3]	A three byte ASCII code that identifies the *Message* type. The following values are defined at this time: MSG A *Message* secured with the keys associated with a c OPN OpenSecureChannel *Message*. CLO CloseSecureChannel *Message*.
IsFinal	Byte	A one byte ASCII code that indicates wh the *MessageC* The following values are defined at time: C An intermediate chunk. F The final chunk. A The final chunk (used when an error occurred and the *Message* is aborted). This field is only meaningful for MessageType of 'MSG' This field is always 'F' for other MessageTypes.



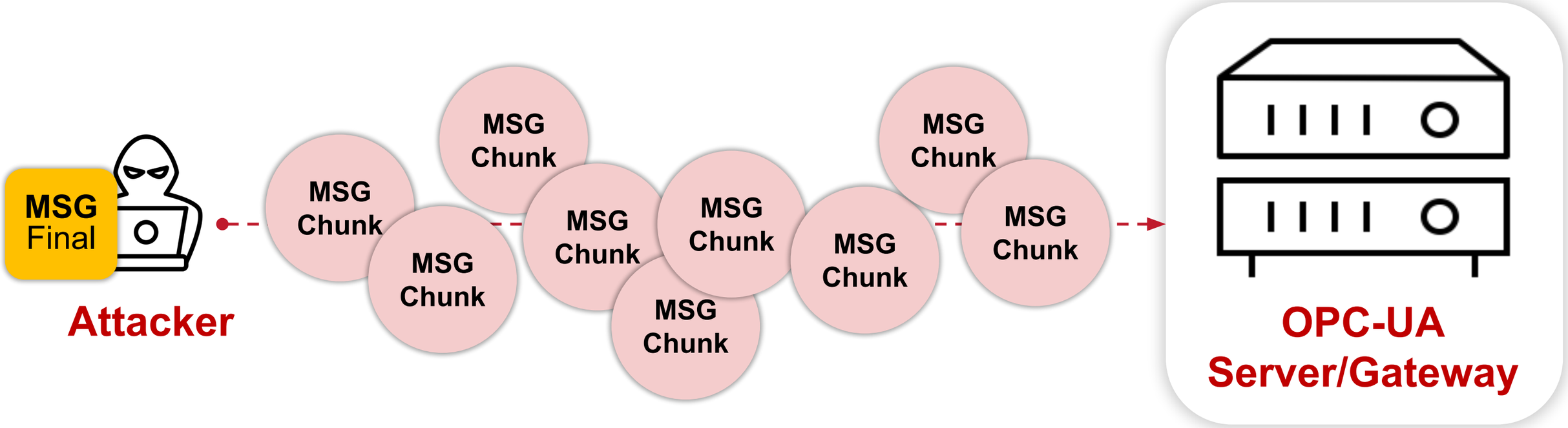
Denial of Service - Chunk Flooding

```
private bool ProcessRequestMessage(uint messageType, ArraySegment<byt
{
        ...
        ...

    try
    {
        // check for an abort.
        if (TcpMessageType.IsAbort(messageType))
        {
            Utils.Trace("Request was aborted.");
            chunksToProcess = GetSavedChunks(requestId, messageBody);
            return true;
        }

        // check if it is necessary to wait for more chunks.
        if (!TcpMessageType.IsFinal(messageType))
        {
            SaveIntermediateChunk(requestId, messageBody);
            return true;
        }
```

OPC-UA .NET Stack

Denial of Service - Chunk Flooding

```
while !isFinalChunk:
    add(chunk)
```



Denial of Service - Chunk Flooding

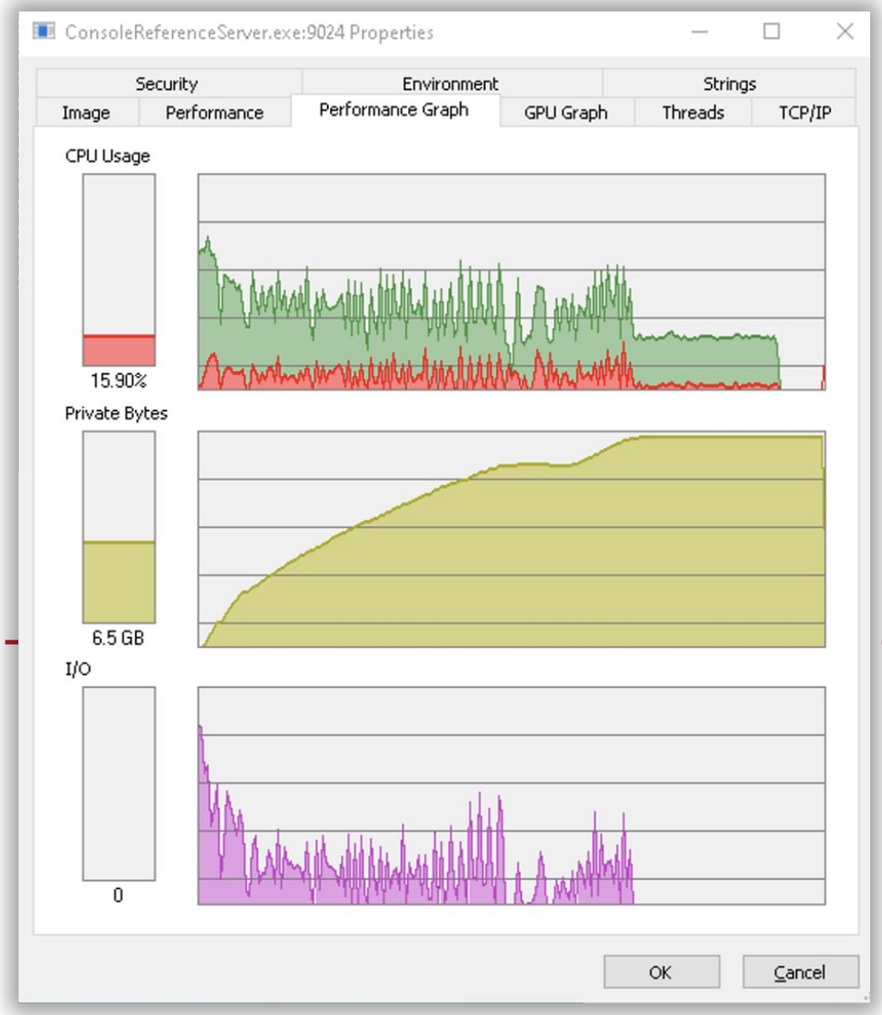
```
while !isFinalChunk:
    add(chunk)
```

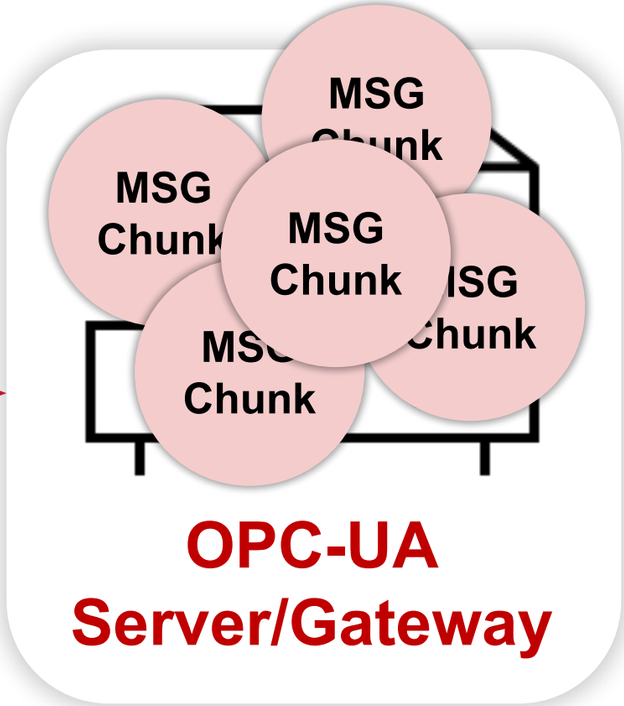


Denial of Service - Chunk Flooding



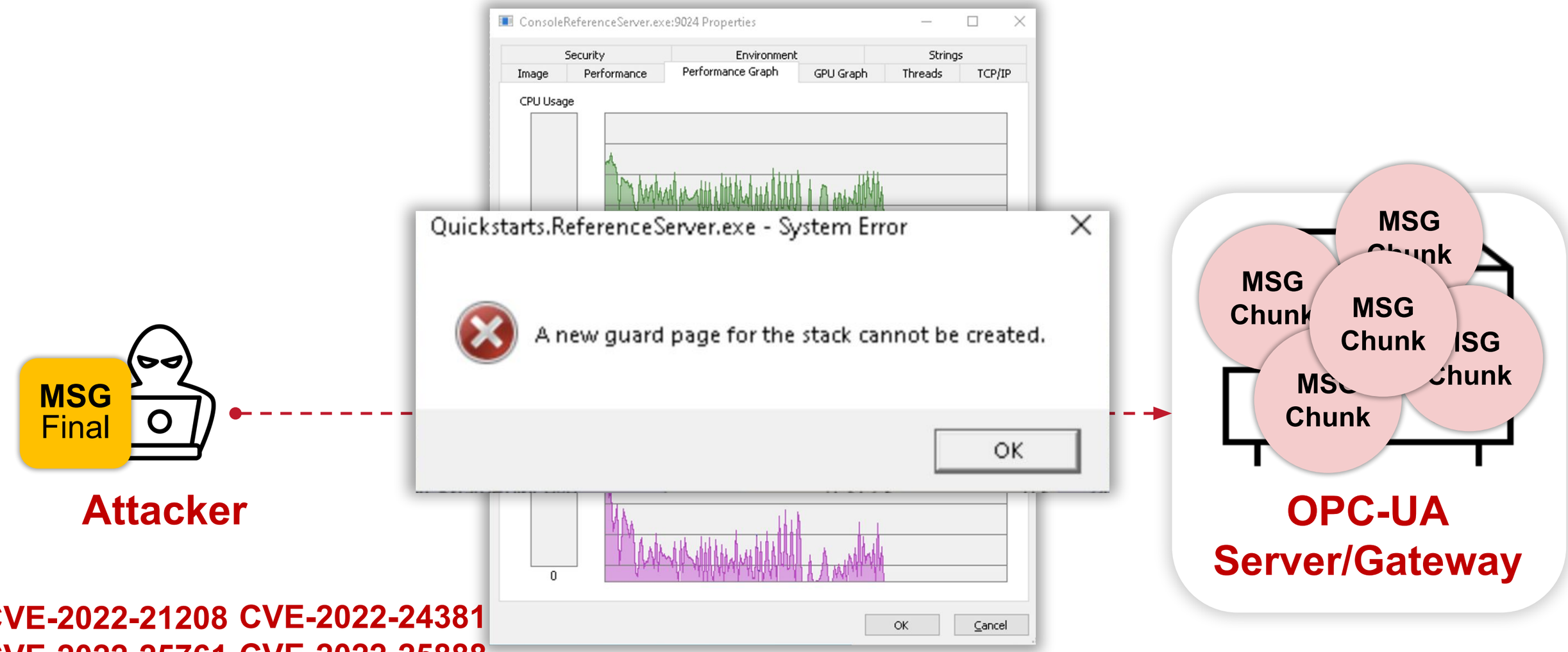
Attacker





OPC-UA
Server/Gateway

Denial of Service - Chunk Flooding



CVE-2022-21208 CVE-2022-24381
CVE-2022-25761 CVE-2022-25888
CVE-2022-25304 CVE-2022-29864

Denial of Service – Attack Concepts

Resource exhaustion - uncontrolled memory management

- Chunk Flooding
- Unlimited ConditionRefresh Attack
- Unlimited Persistent Monitored Subscriptions
- Unlimited Open Channels

Threads deadlock

- Worker Starvation

Use-after-free bugs

- **Method Calling From Dead Session**
- Add/Remove From Namespace While Browsing

Buffer overflows - heap/stack corruption

- Unicode Conversion - OOB Write

Uncaught exceptions

- Parser Bug - Dissecting Malformed OPC-UA Data Type

Busy loops / unlimited recursions – call-stack overflow

- Complex Deep Nested Variants (OTORIO)
- Certificate Chain Loop (Sector7)
- Unlimited Translate Browse Path (JFrog)

Denial of Service - Method Calling From Dead Session

5 Service Sets

5.11 Method Service Set

5.11.2 Call

5.11.2.1 Description

```
@uamethod
def multiply(parent, x, y):
    print("multiply method call with parameters: ", x, y)
    return x * y
```

Example to exposed function (python-opcua)

This *Service* is used to call (invoke) a list of *Methods*.

This *Service* provides for passing input and output arguments to/from a *Method*. These arguments are defined by *Properties* of the *Method*.

If the *Method* is invoked in the context of a *Session* and the *Session* is terminated, the results of the *Method's* execution cannot be returned to the *Client* and are discarded. This is independent of the task actually performed at the *Server*.

The order the operations are processed in the *Server* is not defined and depends on the different tasks and the internal *Server* logic. If a *Method* is contained in more than one operation, the order of the processing is undefined. If a *Client* requires sequential processing the *Client* needs separate *Service* calls.

https://reference.opcfoundation.org/v104/Core/docs/Part4/5.11.2/

Denial of Service - Method Calling From Dead Session

5 Service Sets ↑ ⊖ ⊖

5.11 Method Service Set ↑ ⊖ ⊖

5.11.2 Call ↑ ⊖ ⊖

5.11.2.1 Description ↑

```python
@uamethod
def multiply(parent, x, y):
    print("multiply method call with parameters: ", x, y)
    return x * y
```

Example to exposed function (python-opcua)

This *Service* is used to call (invoke) a list of *Methods*.

This *Service* provides for passing input and output arguments to/from a *Method*. These arguments are defined by *Properties* of the *Method*.

If the *Method* is invoked in the context of a *Session* and the *Session* is terminated, the results of the *Method's* execution cannot be returned to the *Client* and are discarded. This is independent of the task actually performed at the *Server*.

The order the operations are processed in the *Server* is not defined and depends on the different tasks and the internal *Server* logic. If a *Method* is contained in more than one operation, the order of the processing is undefined. If a *Client* requires sequential processing the *Client* needs separate *Service* calls.

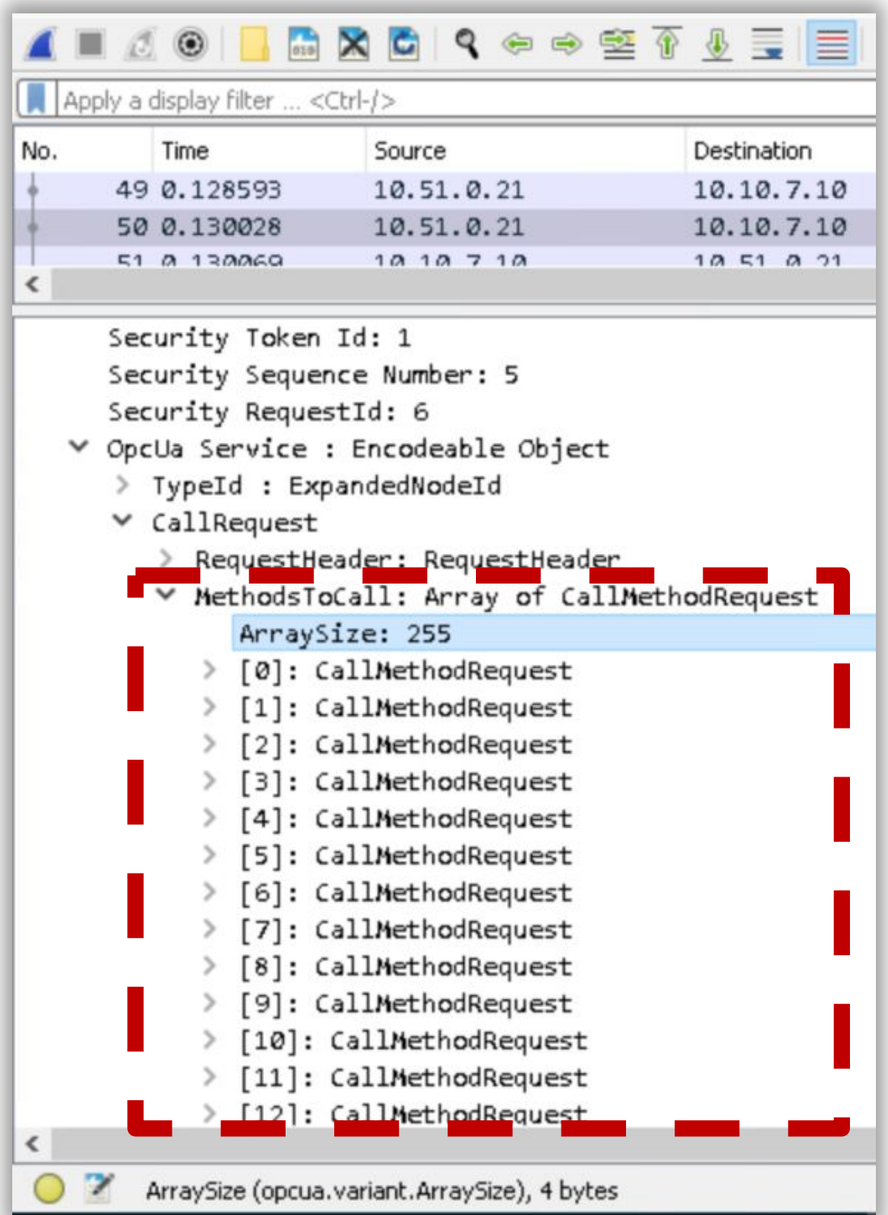
https://reference.opcfoundation.org/v104/Core/docs/Part4/5.11.2/

Denial of Service - Method Calling From Dead Session

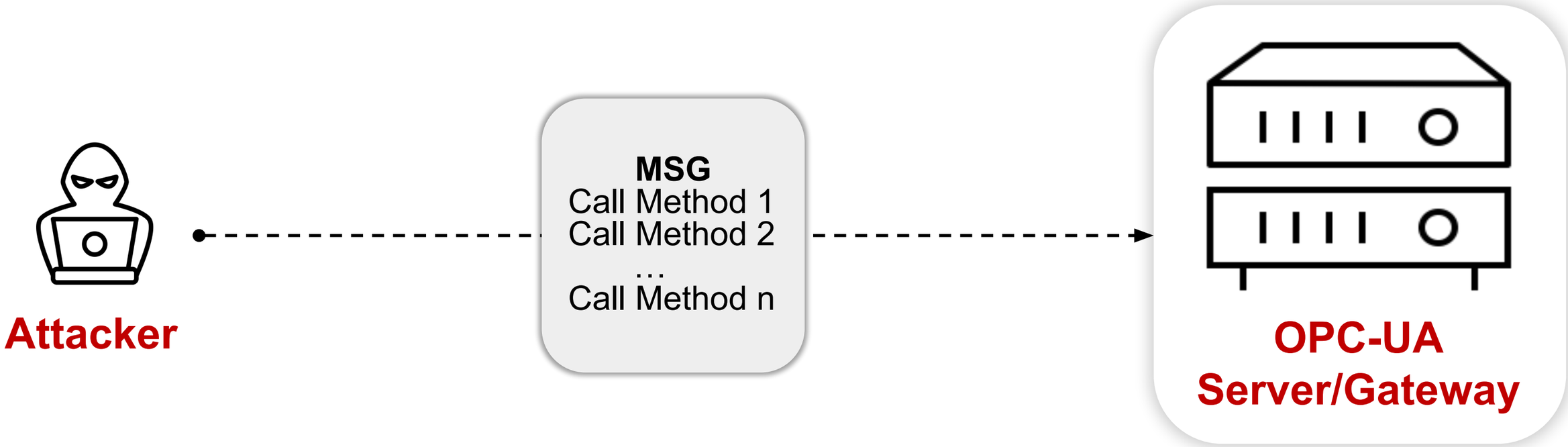
Did all stacks implement this correctly?

Exploit:

- Sending many Call Method Request
- And immediately close the session



Denial of Service - Method Calling From Dead Session



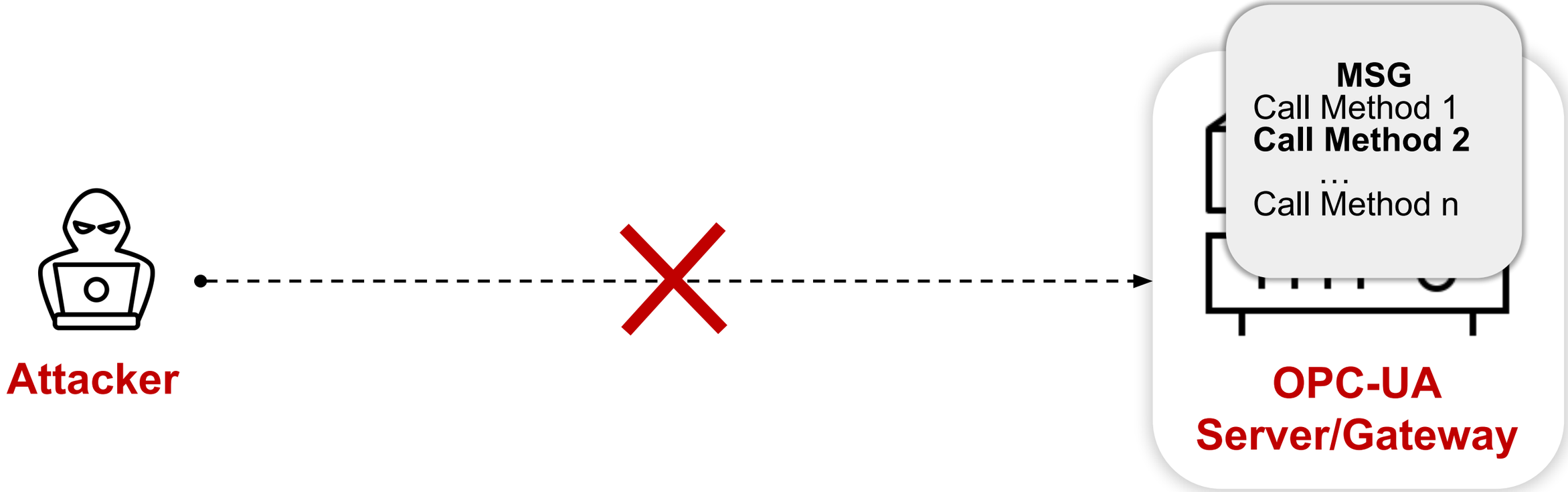
Denial of Service - Method Calling From Dead Session



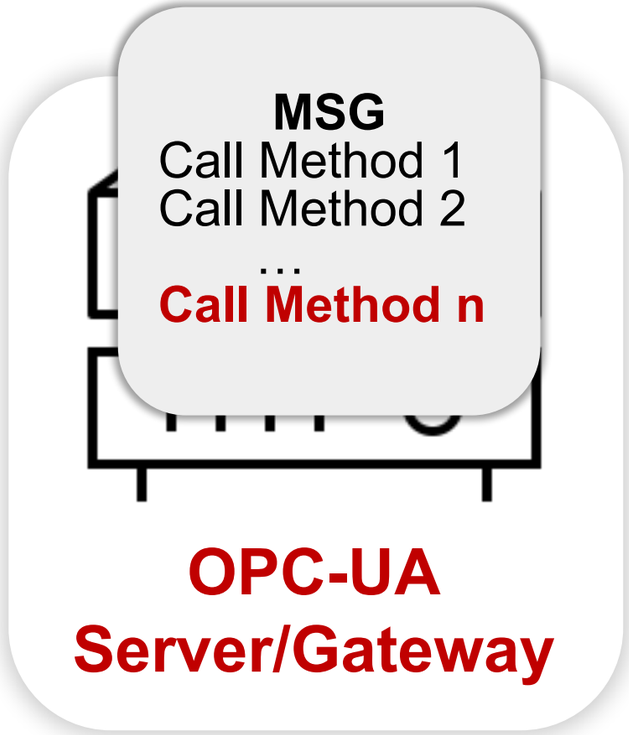
Denial of Service - Method Calling From Dead Session



Denial of Service - Method Calling From Dead Session



Denial of Service - Method Calling From Dead Session



Denial of Service - Method Calling From Dead Session

MSG
Call Method 1
Call Method 2
…
Call Method n

Softing Secure Integration Server

```
OTServerMethodCallRequest::setInputReturns_API((int)a3, 1, &src);
OTServerDataTransaction::getSession(
  (OTServerDataTransaction *)&OTServerDataTransaction,
  OTServerMethodCallTransaction);
```

Denial of Service - Method Calling From Dead Session

MSG
Call Method 1
Call Method 2
…
Call Method n

Softing Secure Integration Server

```
OTServerMethodCallRequest::setInputReturns_API((int)a3, 1, &src);
OTServerDataTransaction::getSession(
  (OTServerDataTransaction *)&OTServerDataTransaction,
  OTServerMethodCallTransaction);
```

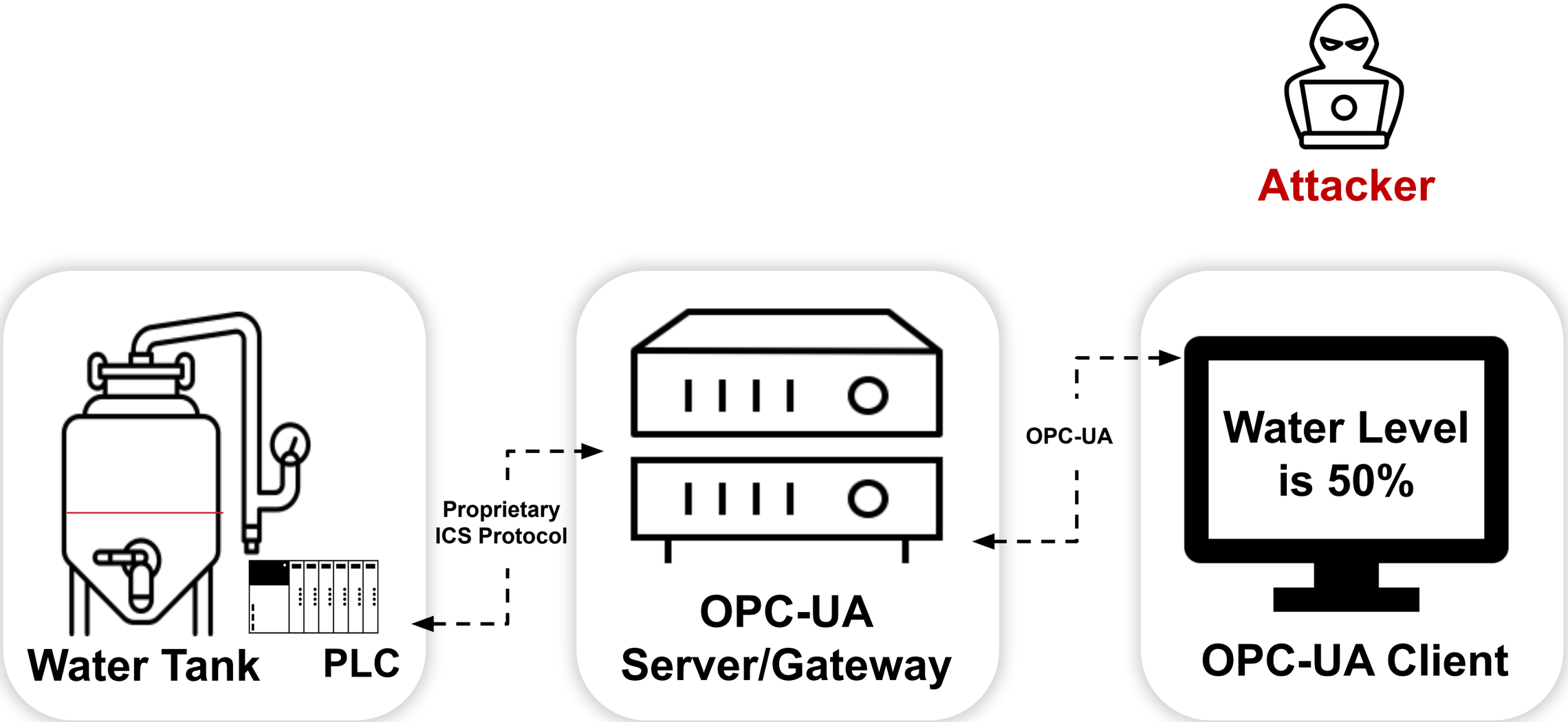
```
(d24.1740): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
ntdll!RtlEnterCriticalSection+0xd:
00007ff9`1529faad f00fba710800    lock btr dword ptr [rcx+8],0 ds:00000000`00000028=????????
```

CVE-2022-1748

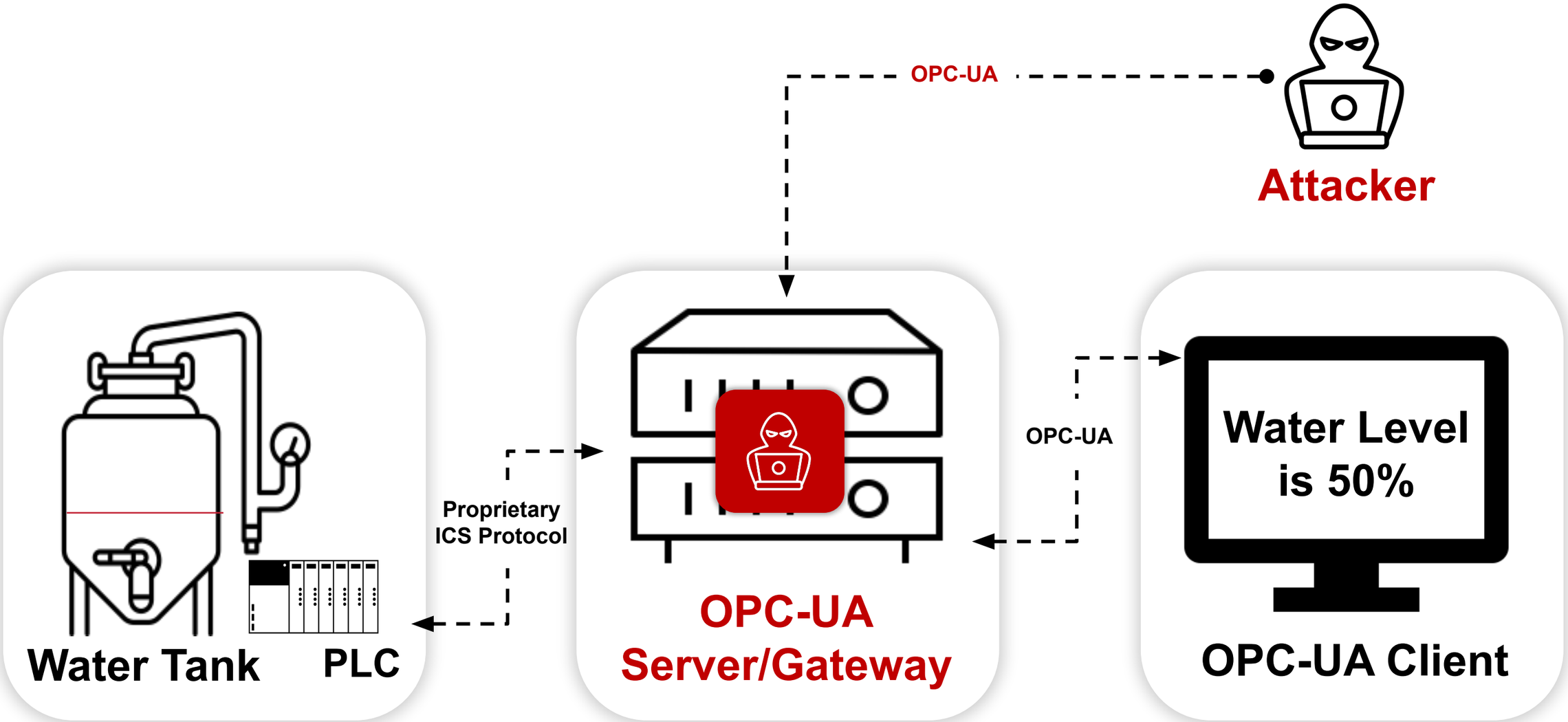
Vulnerabilities and Exploits

RCE - Servers

OPC-UA Server - RCE

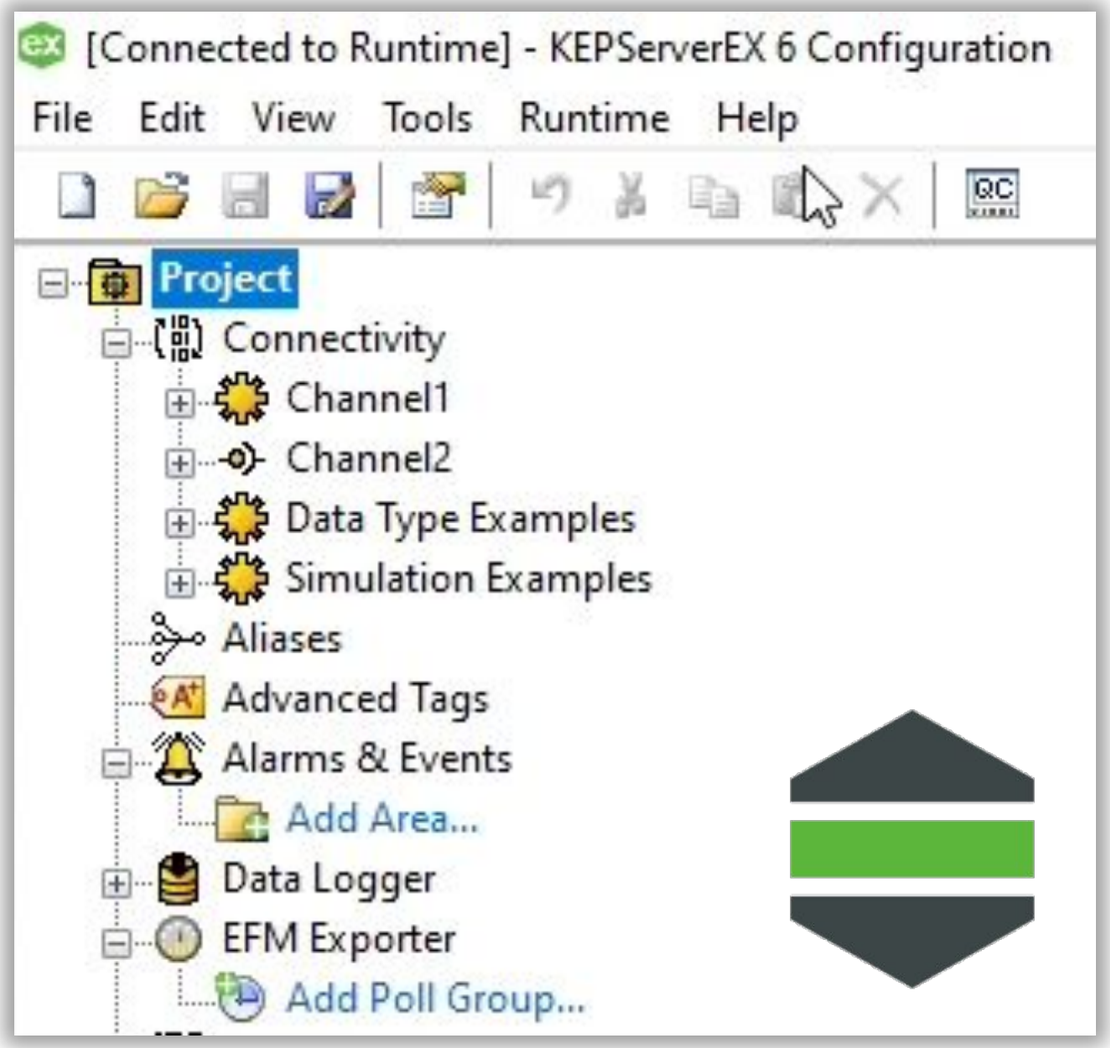


OPC-UA Server - RCE

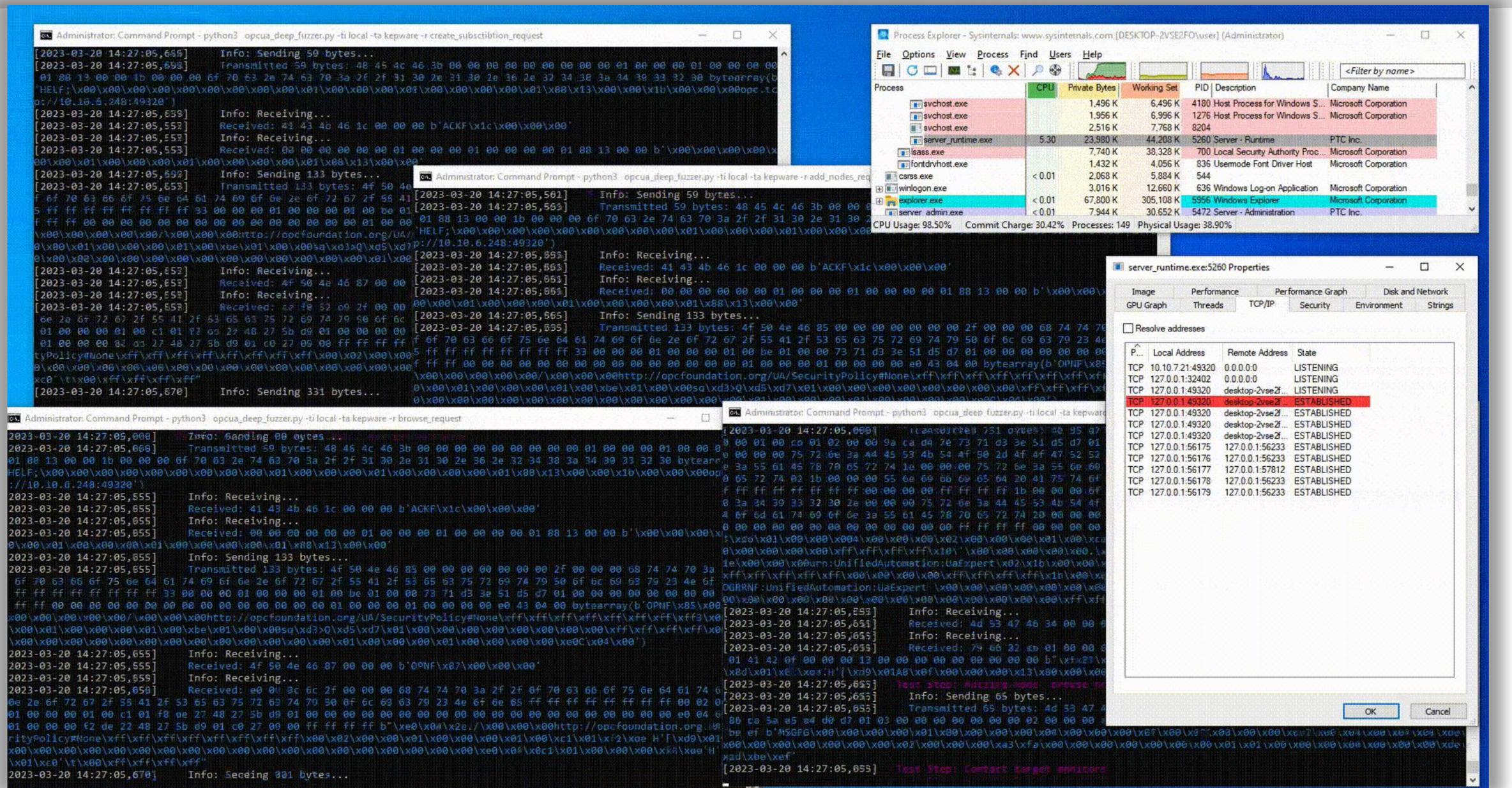


PTC Kepware KepServerEx

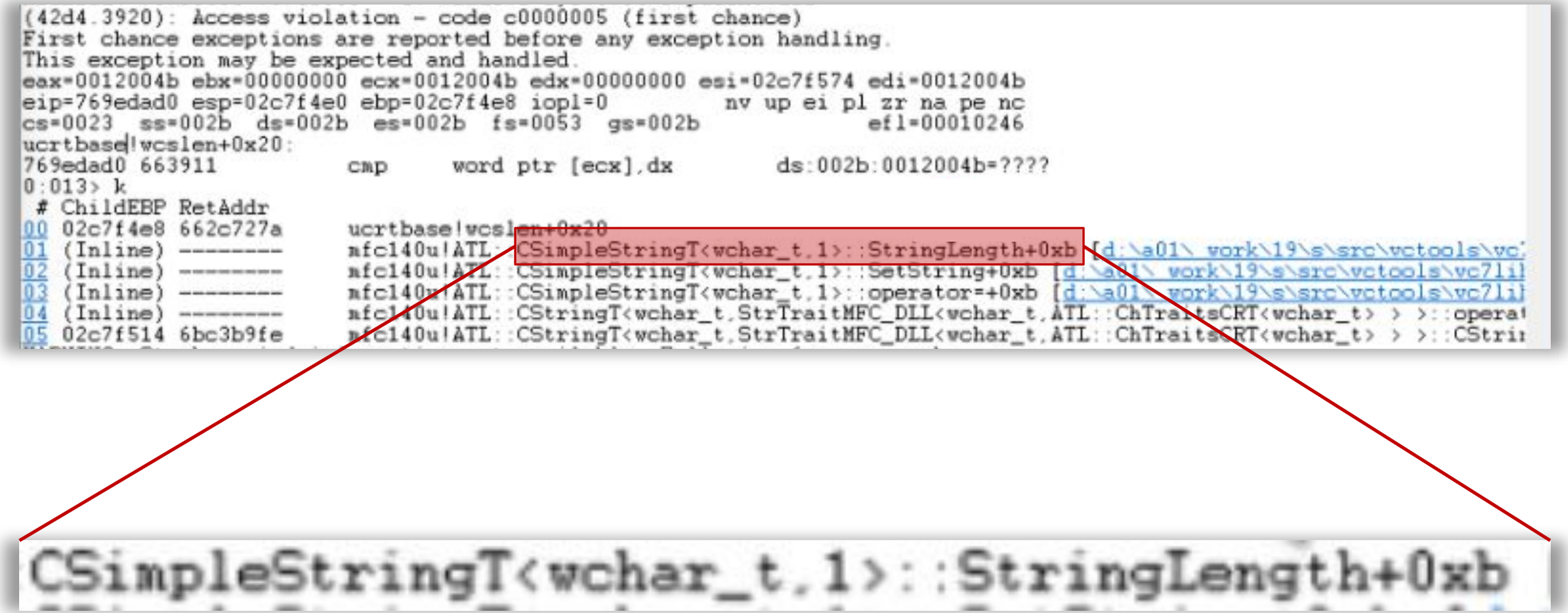
- Industry's leading OPC-UA server, used in biggest manufacturing lines, oil rigs, wind farms, etc.
- Windows-based
- Custom OPC-UA protocol stack
- OPC-UA logic in `server_runtime.exe`
  - 32bit, service (SYSTEM)
  - Customized anti-debugging



Fuzzer Demo

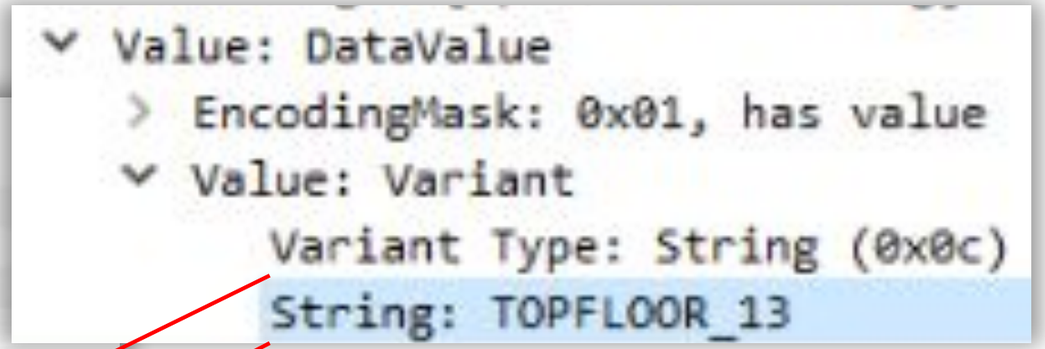

Administrator: Command Prompt - python3 opcua_deep_fuzzer.py -ti local -ta kepware -r create_subsctibtion_request
Administrator: Command Prompt - python3 opcua_deep_fuzzer.py -ti local -ta kepware -r add_nodes_req
Administrator: Command Prompt - python3 opcua_deep_fuzzer.py -ti local -ta kepware -r browse_request
Administrator: Command Prompt - python3 opcua_deep_fuzzer.py -ti local -ta kepware
Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-2VSE2FO\user] (Administrator)
File Options View Process Find Users Help
Process CPU Private Bytes Working Set PID Description Company Name
svchost.exe 1,496 K 6,496 K 4180 Host Process for Windows S... Microsoft Corporation
svchost.exe 1,956 K 6,996 K 1276 Host Process for Windows S... Microsoft Corporation
svchost.exe 2,516 K 7,768 K 8204
server_runtime.exe 5.30 23,980 K 44,208 K 5260 Server - Runtime PTC Inc.
lsass.exe 7,740 K 38,328 K 700 Local Security Authority Proc... Microsoft Corporation
fontdrvhost.exe 1,432 K 4,056 K 836 Usermode Font Driver Host Microsoft Corporation
csrss.exe < 0.01 2,068 K 5,884 K 544
winlogon.exe 3,016 K 12,660 K 636 Windows Log-on Application Microsoft Corporation
explorer.exe < 0.01 67,800 K 305,108 K 5956 Windows Explorer Microsoft Corporation
server_admin.exe < 0.01 7,944 K 30,652 K 5472 Server - Administration PTC Inc.
CPU Usage: 98.50% Commit Charge: 30.42% Processes: 149 Physical Usage: 38.90%
server_runtime.exe:5260 Properties
Image Performance Performance Graph Disk and Network
GPU Graph Threads TCP/IP Security Environment Strings
Resolve addresses
P... Local Address Remote Address State
TCP 10.10.7.21:49320 0.0.0.0:0 LISTENING
TCP 127.0.0.1:32402 0.0.0.0:0 LISTENING
TCP 127.0.0.1:49320 desktop-2vse2f... LISTENING
TCP 127.0.0.1:49320 desktop-2vse2f... ESTABLISHED
TCP 127.0.0.1:49320 desktop-2vse2f... ESTABLISHED
TCP 127.0.0.1:49320 desktop-2vse2f... ESTABLISHED
TCP 127.0.0.1:49320 desktop-2vse2f... ESTABLISHED
TCP 127.0.0.1:56175 127.0.0.1:56233 ESTABLISHED
TCP 127.0.0.1:56176 127.0.0.1:56233 ESTABLISHED
TCP 127.0.0.1:56177 127.0.0.1:57812 ESTABLISHED
TCP 127.0.0.1:56178 127.0.0.1:56233 ESTABLISHED
TCP 127.0.0.1:56179 127.0.0.1:56233 ESTABLISHED
OK
Cancel

Analyzing the Crash

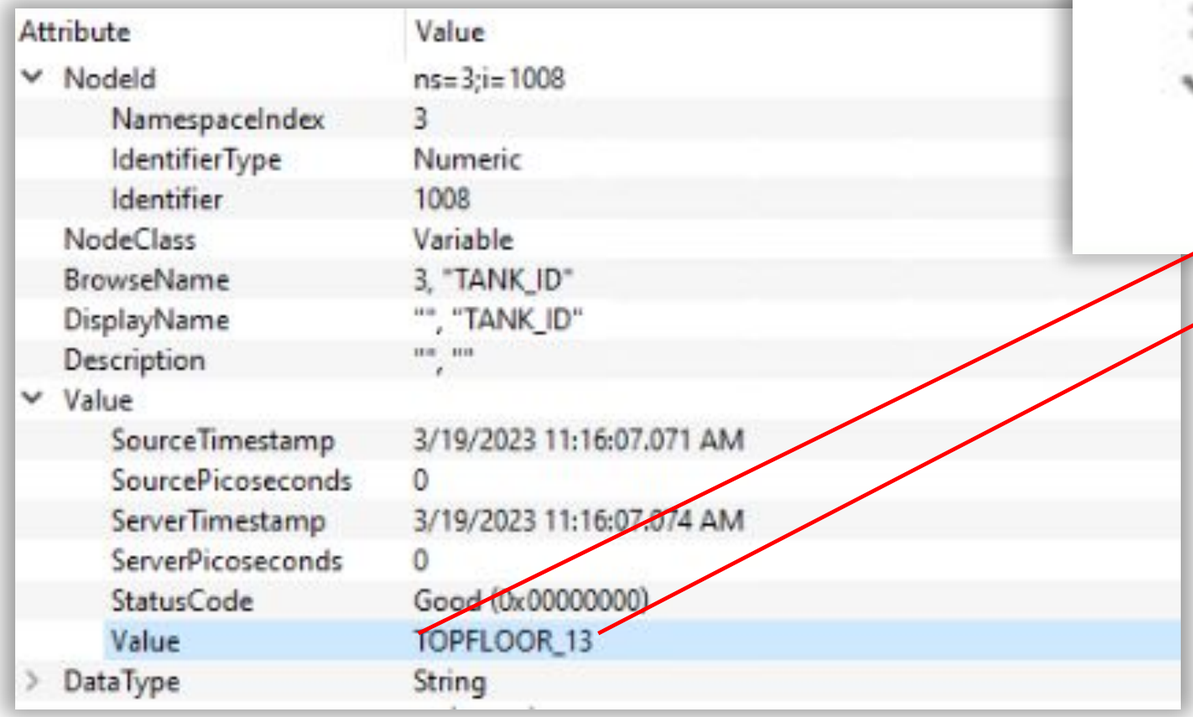


CSimpleStringT<wchar_t,1>::StringLength+0xb

OPC-UA Strings are UTF-8 Encoded



Read tag's value Wireshark



TANK_ID tag and it's value
Unified Automation Client

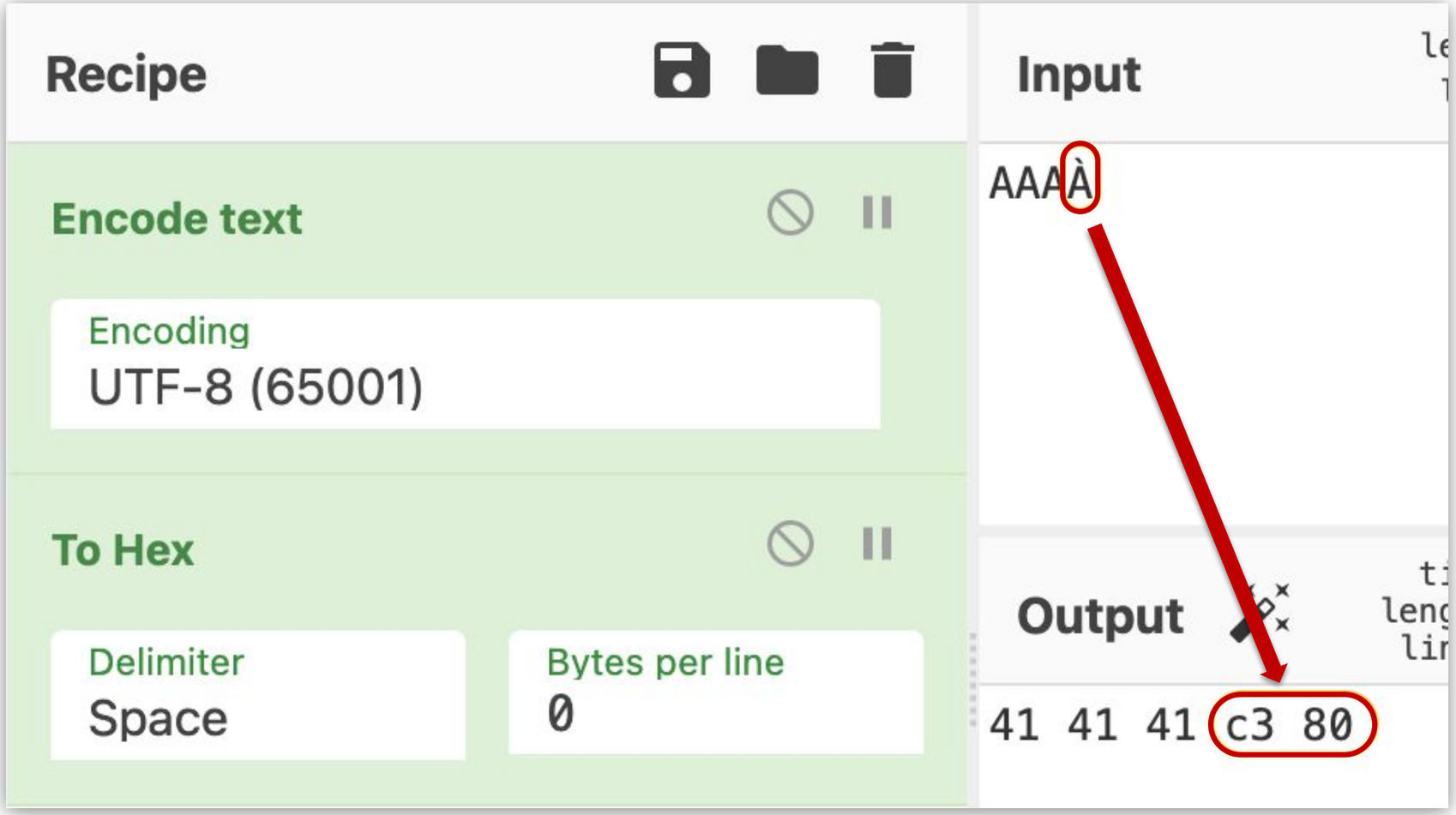
KepServerEx Conversion bug

KepServerEx is *trying* to convert UTF-8 to UTF-16

```
1struct_a1 *__cdecl CUtf8String::ToWide(struct_a1 *struct_for_cstring, char *input_string)
2{
3  unsigned int number_of_utf16_code_units; // eax
4  struct_a1 *result; // eax
5
6  number_of_utf16_code_units = CUtf8String::GetUtf16Length(input_string);
7  struct_for_cstring->number_of_utf16_code_units = 0;
8  struct_for_cstring->lenght = 7;
9  LOWORD(struct_for_cstring->pointer_to_heap_allocated_buffer) = 0;
10  PERFORM_ALLOCATIONS(struct_for_cstring, number_of_utf16_code_units, 0);
11  result = struct_for_cstring;
12  if ( !struct_for_cstring->number_of_utf16_code_units )
13    return result;
14  if ( struct_for_cstring->lenght >= 8u )
15    result = (struct_a1 *)struct_for_cstring->pointer_to_heap_allocated_buffer;
16  CUtf8String::ToWide(input_string, (char *)result);
17  result = struct_for_cstring;
18  return result;
19}
```

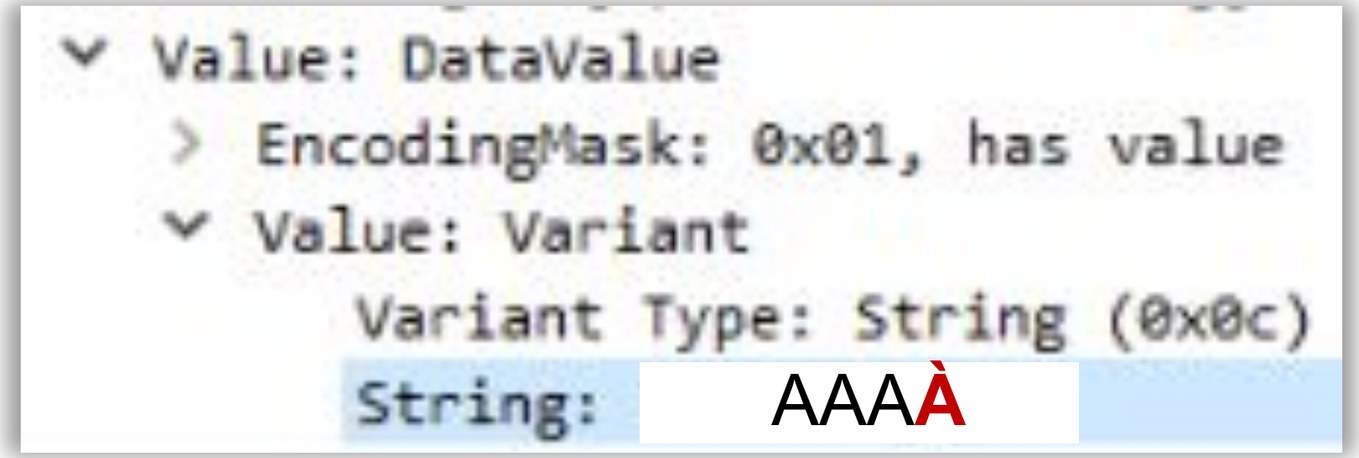
```
number_of_utf16_code_units = CUtf8String::GetUtf16Length(input_string);
```

String Encoding is Hard



UTF-8 to UTF-16 is Hard

UTF-8: AAA**À**



Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA**À**



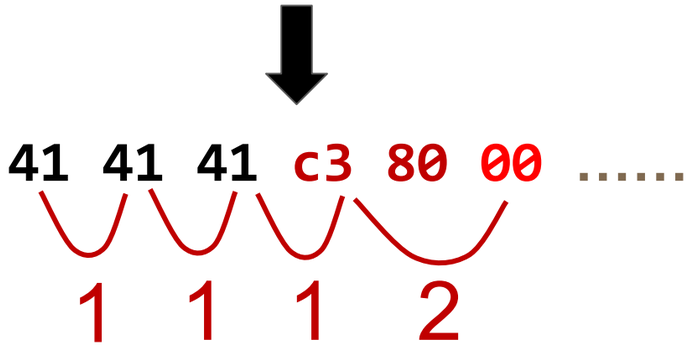
```
41 41 41 c3 80 00 ......
```

Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAAÀ

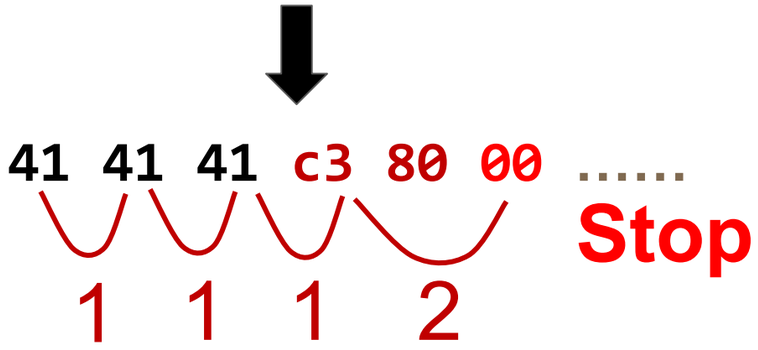


Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAAÀ

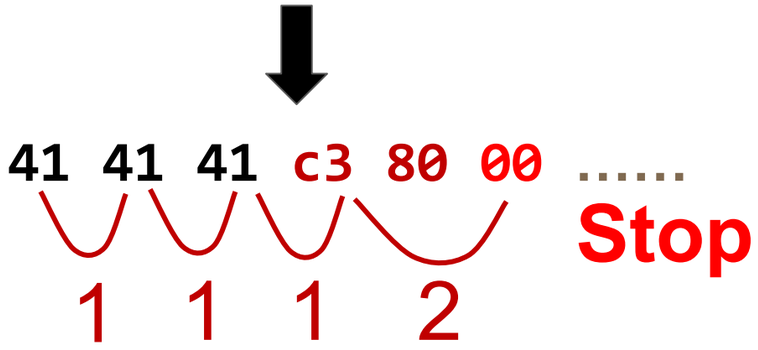


Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA**À**



UTF-16:



\x41\x41\x41\xC3\x80

Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA**À**

Whatever starts with C3 is

OK

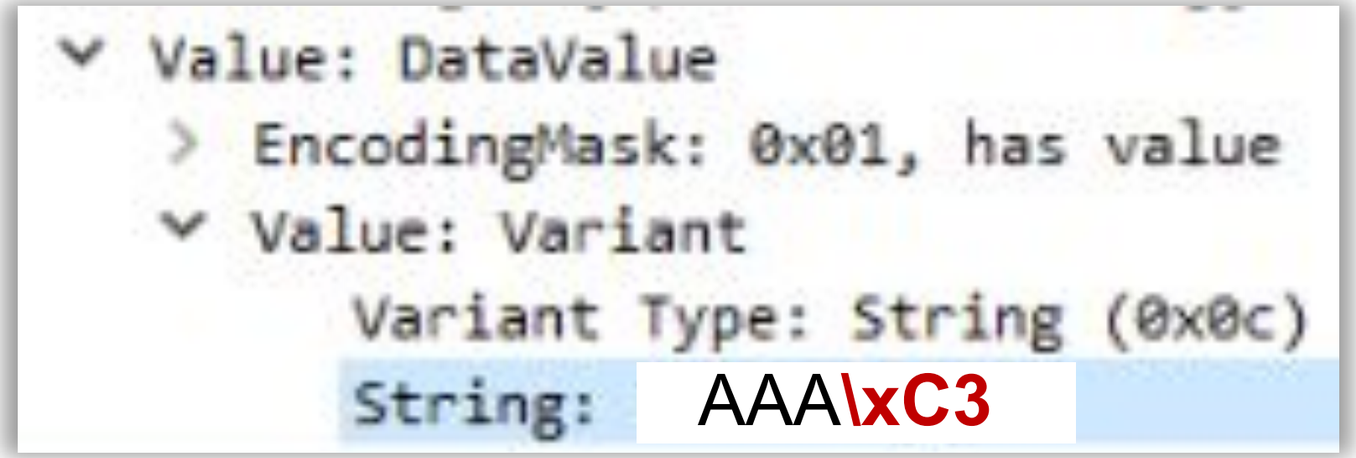


\x41\x41\x41\xC3\x80



UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3



Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA**\xC3**



```
41 41 41 c3 00
```

Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3



```
41 41 41 c3 00
```

1

Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3



```
41 41 41 c3 00
```

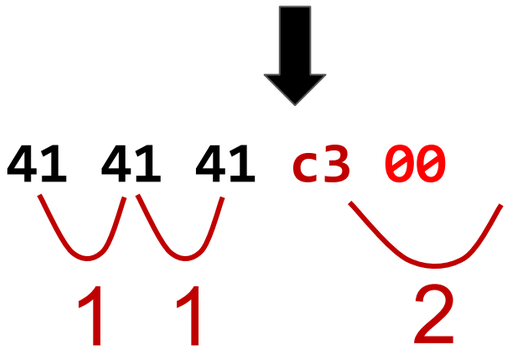
1 1

Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3

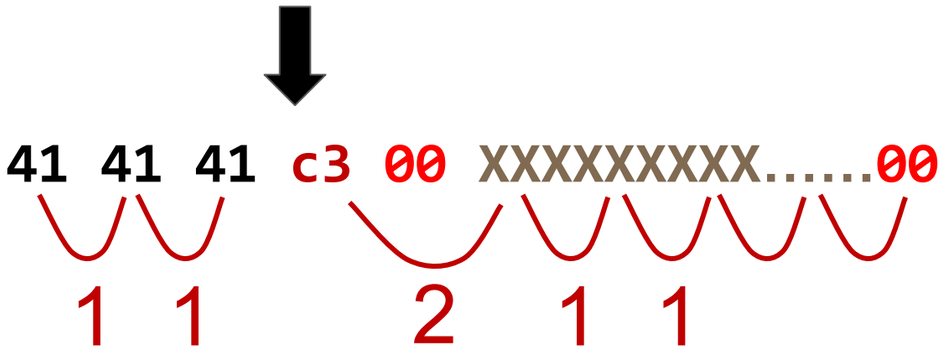


Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3

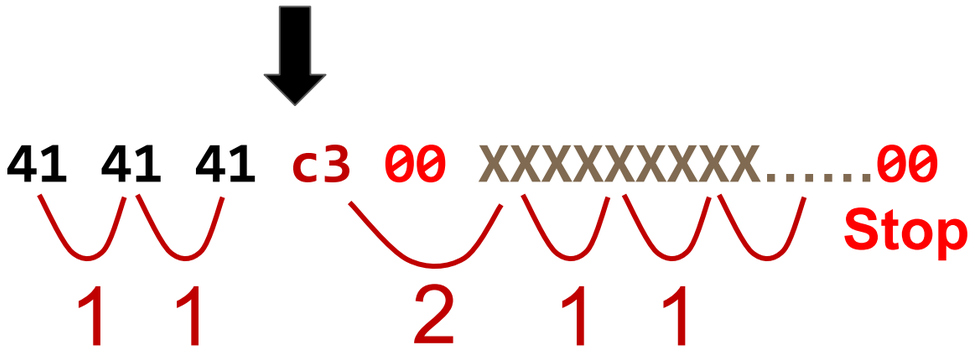


Whatever starts with C3 is probably 2 bytes long



UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3

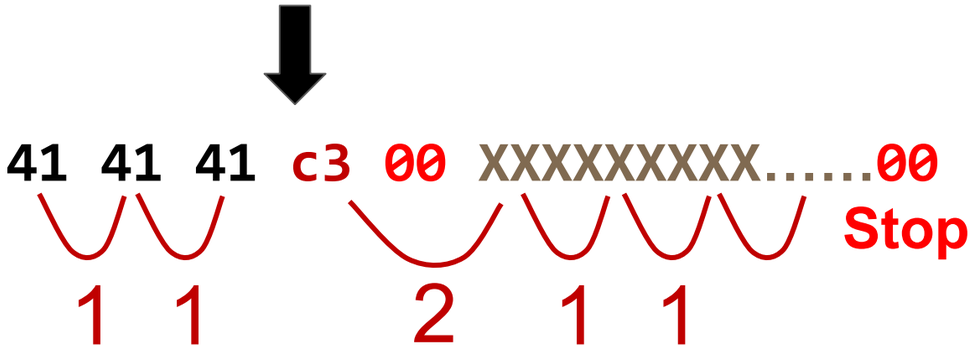


Whatever starts with C3 is probably 2 bytes long

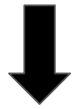


UTF-8 to UTF-16 is Hard

UTF-8: AAA**\xC3**



UTF-16:

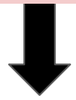


\x41\x41\x41\xC3LEAKINGTHEHEAP

UTF-8 to UTF-16 is Hard

UTF-8: AAA\xC3

FAIL



\x41\x41\x41\xC3LEAKINGTHEHEAP

Heap Overflow Primitive

The bug is triggered on both READ_TAG and WRITE_TAG functions

We have heap OOB (read+write)

- OOB read → leak pointers to defeat ASLR
- OOB write → construct ROP chain, RCE and PWN

Heap OOB Read

```
▾ OpcUa Service : Encodeable Object
  › TypeId : ExpandedNodeId
  ▾ ReadResponse
    › ResponseHeader: ResponseHeader
    ▾ Results: Array of DataValue
        ArraySize: 1
      ▾ [0]: DataValue
        › EncodingMask: 0x0d, has value, has source timestamp, has server timestamp
        ▾ Value: Variant
            Variant Type: String (0x0c)
            String: aa0h<\U000B5A2C002\;
          SourceTimestamp: Feb 15, 2022 14:29:33.498526100 GMT Standard Time
          ServerTimestamp: Feb 15, 2022 14:29:33.498526100 GMT Standard Time
    › DiagnosticInfos: Array of DiagnosticInfo
```

```
0000  4d 53 47 46 5f 00 00 00  4b 74 a2 6d 01 00 00 00   MSGF_··· Kt·m····
0010  04 02 00 00 05 00 00 00  01 00 7a 02 2d 16 67 73   ········ ··z·-·gs
0020  78 22 d8 01 41 00 00 00  00 00 00 00 00 00 00 00   x"··A··· ········
0030  00 00 00 00 01 00 00 00  0d 0c 0d 00 00 00 61 61   ········ ······aa
0040  df 80 68 3c f2 b5 a8 ac  db 88 02 2d 16 67 73 78   ··h<···· ···-·gsx
0050  22 d8 01 2d 16 67 73 78  22 d8 01 00 00 00 00      "··-·gsx "·····
```

Leaking data via read tag

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF8	→	**UTF16**
mspaint	→	\x00**m**\x00**s**\x00**p**\x00**a**\x00**i**\x00**n**\x00**t**

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF8	→	**UTF16**
mspaint	→	\x00**m**\x00**s**\x00**p**\x00**a**\x00**i**\x00**n**\x00**t**



Not good for our ROP

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF8	→	**UTF16**
`?????`	→	`mspaint`

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF-8	UTF-16
?	ms

UTF8	→	**UTF16**
?????	→	mspaint

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF-8	UTF-16
獱	ms

UTF8	→	**UTF16**
?????	→	mspaint

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF8	→	**UTF16**
?????	→	mspaint

UTF-8	UTF-16
獱	ms
慰	pa
湩	in
..	..

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

To construct the ROP chain we need to control the whole payload.

UTF8	→	**UTF16**
?????	→	mspaint
獭慰湩憖硥e	→	mspaint.exe

UTF-8	UTF-16
獭	ms
慰	pa
湩	in
..	..

OOB Write

$$\text{UTF-8}_{\text{to}}16(?_{\text{UTF-8}}) = \text{‘ms’}$$

OOB Write

$$\text{UTF-8}_{\text{to}}16(?_{\text{UTF-8}}) = \text{‘ms’} \rightarrow \backslash x6d\backslash x73$$

OOB Write

Unicode Character “獭” (U+736D)

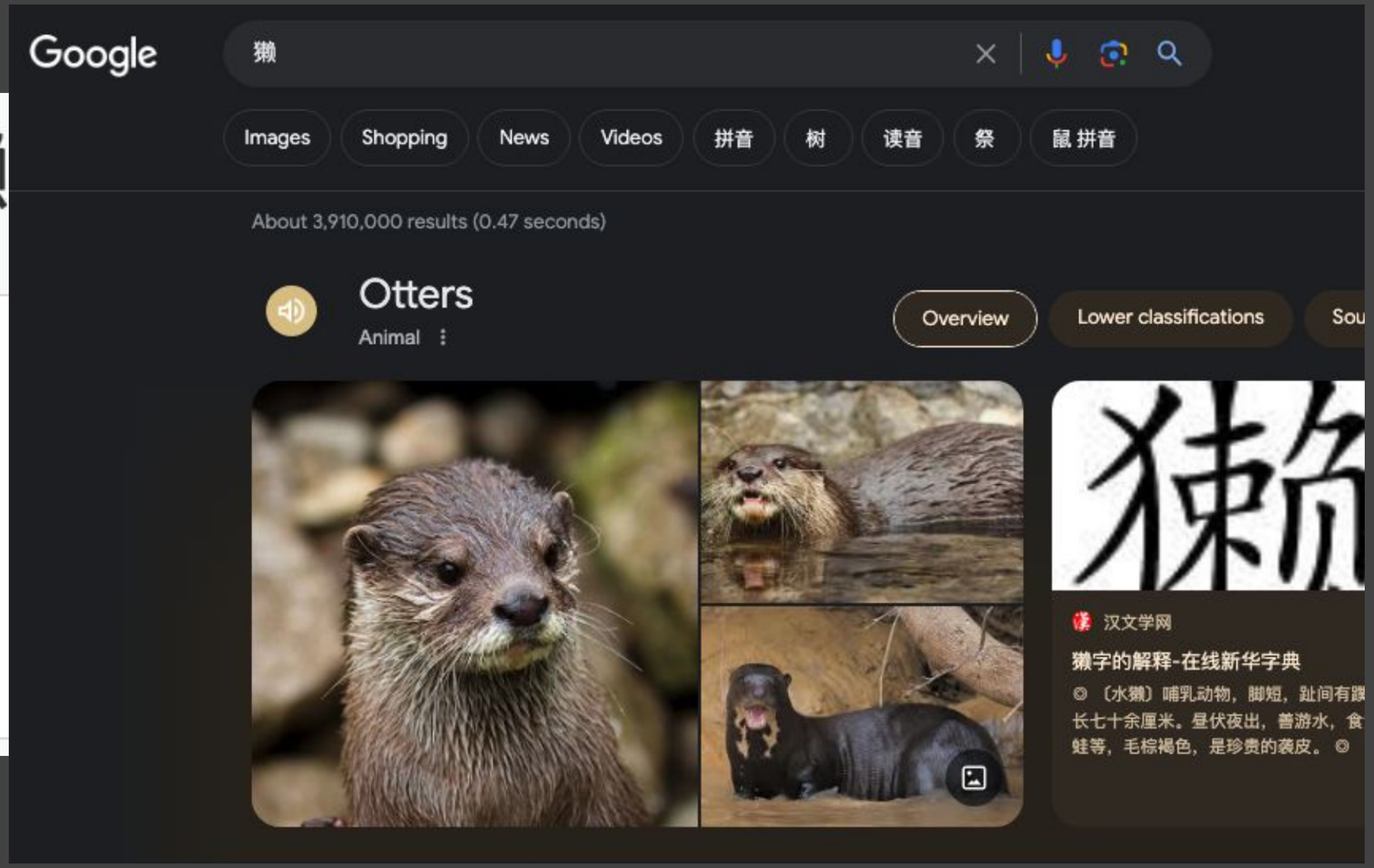


→ \x6d\x73

OOB Write

Unicode Character “獺

獺



OOB Write

$$\text{UTF-8}_{\text{to}}\text{16}(?_{\text{UTF-8}}) = \text{‘ms’} \rightarrow \backslash x6d\backslash x73$$

$$\text{Unicode}(\backslash x6d\backslash x73_{\text{UTF-16}}) = \text{獭}$$

OOB Write

UTF-8$_{to}$16(?$_{UTF-8}$) = ‘ms’ → \x6d\x73

Unicode(\x6d\x73$_{UTF-16}$) = 獭

UTF-8(獭) = \xe7\x8d\xad

OOB Write

UTF-8$_{to}$16(?$_{UTF-8}$) = ‘ms’ → \x6d\x73

Unicode(\x6d\x73$_{UTF-16}$) = 獭

UTF-8(獭) = \xe7\x8d\xad

UTF-8$_{to}$16(\xe7\x8d\xad$_{UTF-8}$) = ?

OOB Write

$$\text{UTF-8}_{\text{to}}\text{16}(?_{\text{UTF-8}}) = \text{‘ms’} \rightarrow \backslash x6d\backslash x73$$

$$\text{Unicode}(\backslash x6d\backslash x73_{\text{UTF-16}}) = 獭$$

$$\text{UTF-8}(獭) = \backslash xe7\backslash x8d\backslash xad$$

$$\text{UTF-8}_{\text{to}}\text{16}(\backslash xe7\backslash x8d\backslash xad_{\text{UTF-8}}) = \backslash x6d\backslash x73$$

OOB Write

$$\text{UTF-8}_{\text{to}}16(?_{\text{UTF-8}}) = \text{‘ms’} \rightarrow \backslash x6d\backslash x73$$

$$\text{Unicode}(\backslash x6d\backslash x73_{\text{UTF-16}}) = 獭$$

$$\text{UTF-8}(獭) = \backslash xe7\backslash x8d\backslash xad$$

$$\text{UTF-8}_{\text{to}}16(\backslash xe7\backslash x8d\backslash xad_{\text{UTF-8}}) = \backslash x6d\backslash x73 = \text{‘ms’}$$

OOB Write

UTF-8$_{to}$16(?$_{UTF-8}$) = ‘ms’ → \x6d\x73

Unicode(\x6d\x73$_{UTF-16}$) = 獭

UTF-8(獭) = \xe7\x8d\xad

UTF-8$_{to}$16(\xe7\x8d\xad$_{UTF-8}$) = \x6d\x73 = ‘ms’

\xe7\x8d\xad$_{UTF-8}$ → **ms**$_{UTF-16}$

OOB Write

We have the pointers to start our ROP chain

But the bytes written are UTF-16 converted

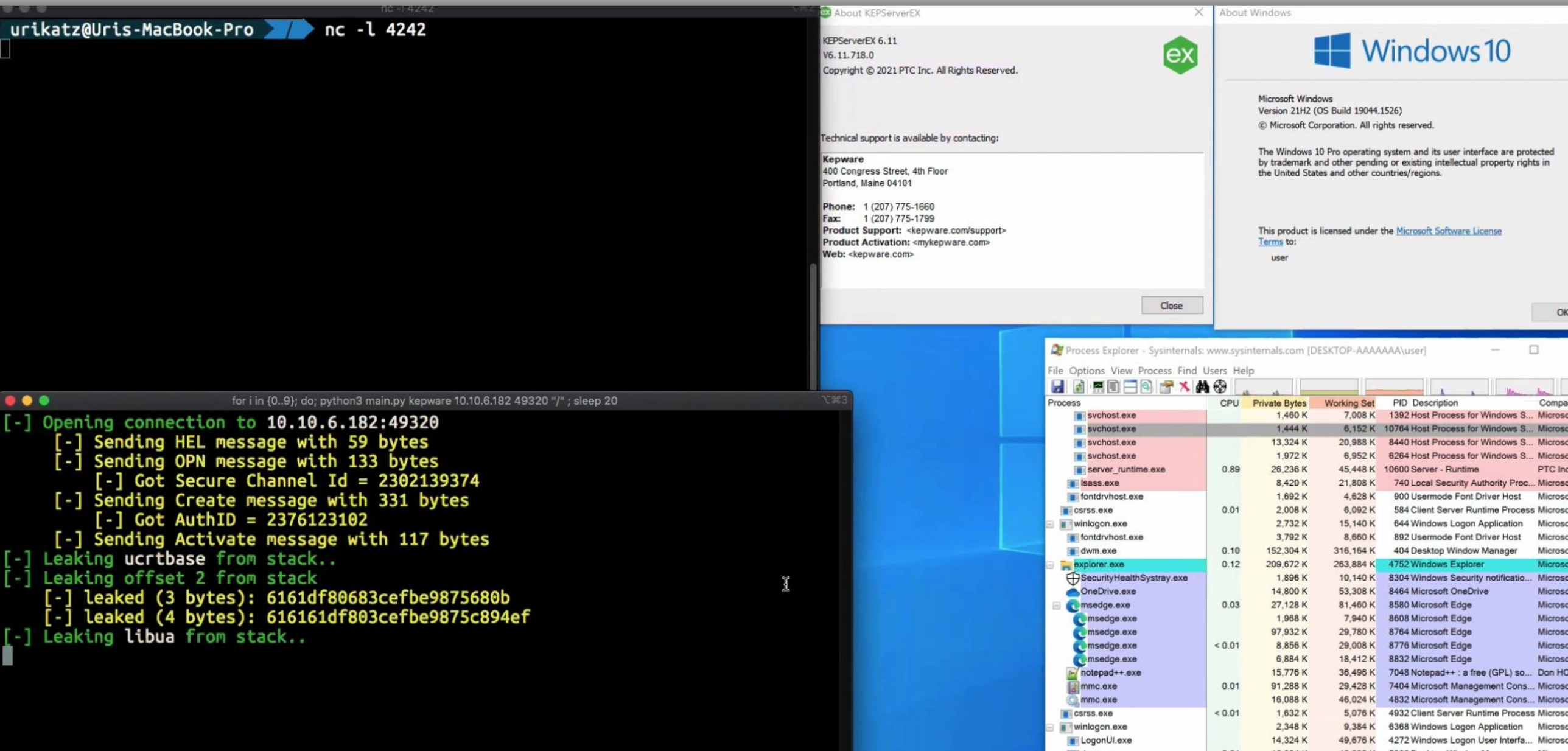
To construct the ROP chain we need to control the whole payload.

```
In [26]: b'mspaint.exe\x00'.decode("utf-16-le").encode("utf-8")
Out[26]: b'\xe7\x8d\xad\xe6\x85\xb0\xe6\xb9\xa9\xe2\xb9\xb4\xe7
```

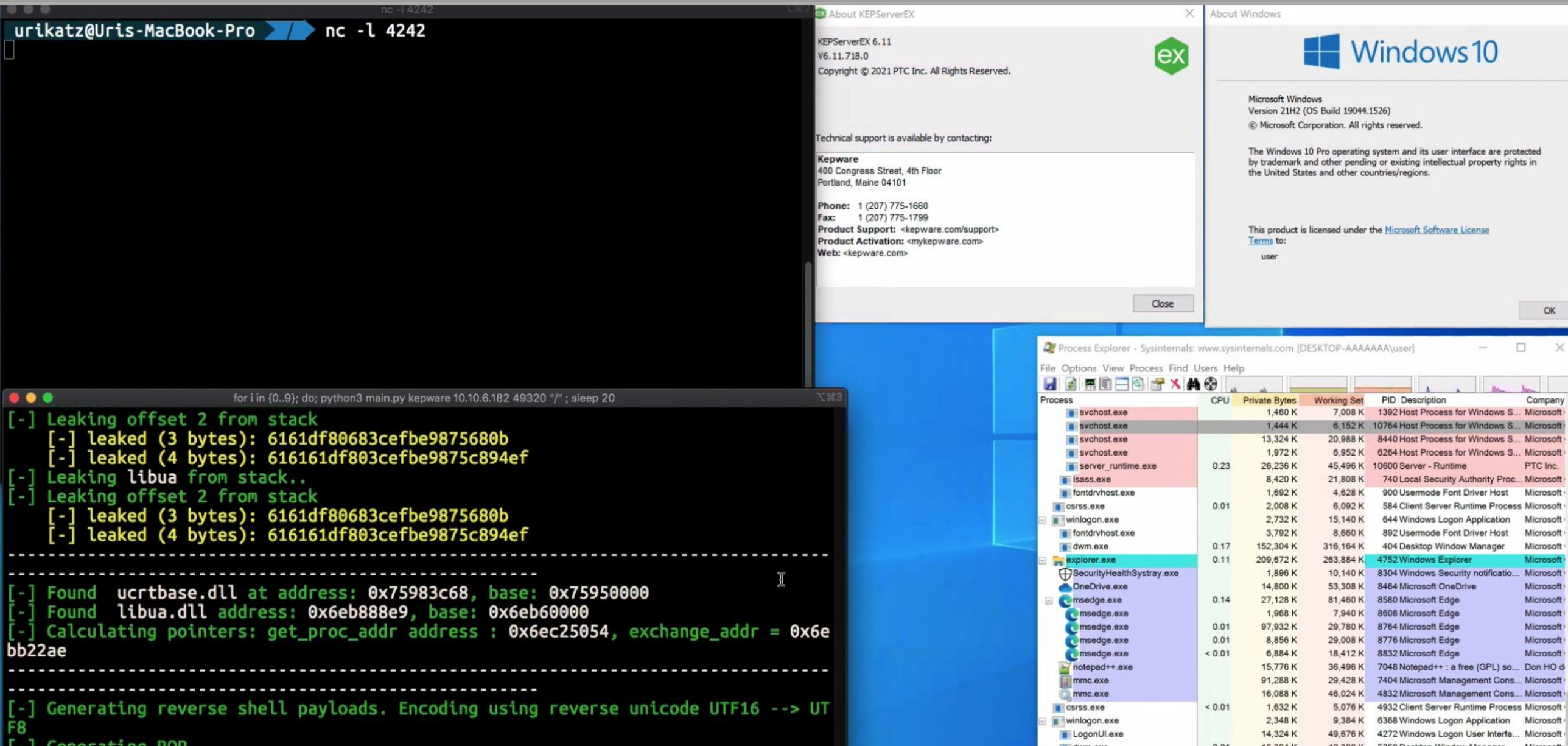
Building the ROP Chain

```
if kep_version == KEPWARE_VERSION_OLD:
    eax = get_encoded_address(libua_base + 0x[redacted])
else:
    eax = get_encoded_address(libua_base + 0x[redacted])
edx = get_encoded_address(ucrt_base + 0x[redacted])
edx += get_encoded_address(libua_base + 0x[redacted])
ebx = get_encoded_address(ucrt_base + 0x[redacted])
ebp = get_encoded_address(libua_base + 0x[redacted])
ebp += get_encoded_address(libua_base + 0x[redacted])
edi = get_encoded_address(ucrt_base + 0x[redacted])
edi += get_encoded_address(libua_base + 0x[redacted])
if kep_version == KEPWARE_VERSION_OLD:
    pushad = get_encoded_address(libua_base + 0x[redacted])
else:
    pushad = get_encoded_address(ucrt_base + 0x[redacted])

mspaint_encoded = b'mspaint.exe\x00'.decode("utf-16-le").encode("utf-8")
mspaint_string = b'\xe7\x8d\xad\xe6\x85\xb0\xe6\xb9\xa9\xe2\xb9\xb4\xe7\xa1\xa5e'
```

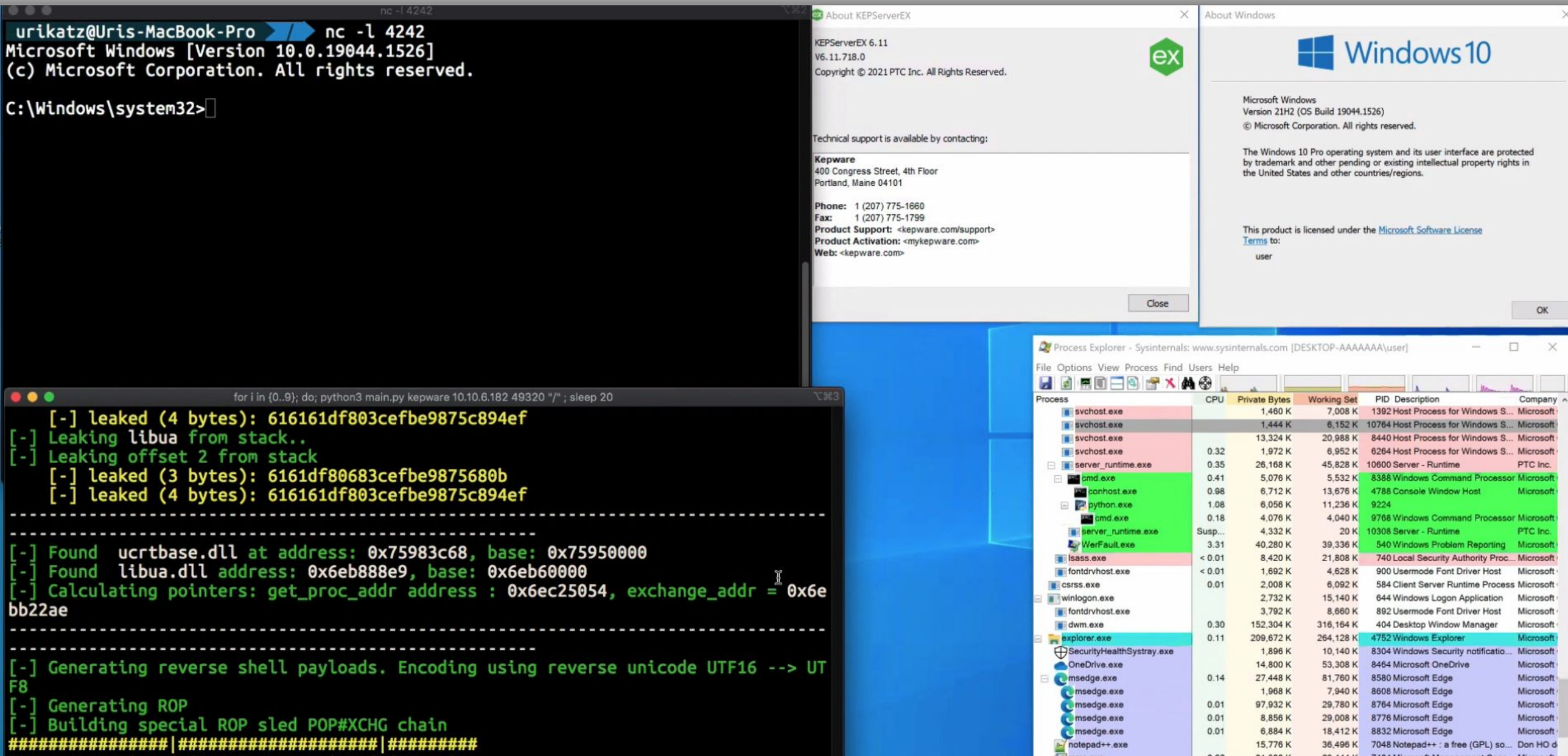
PTC Kepware RCE - Leaking



PTC Kepware RCE - Overwriting Heap

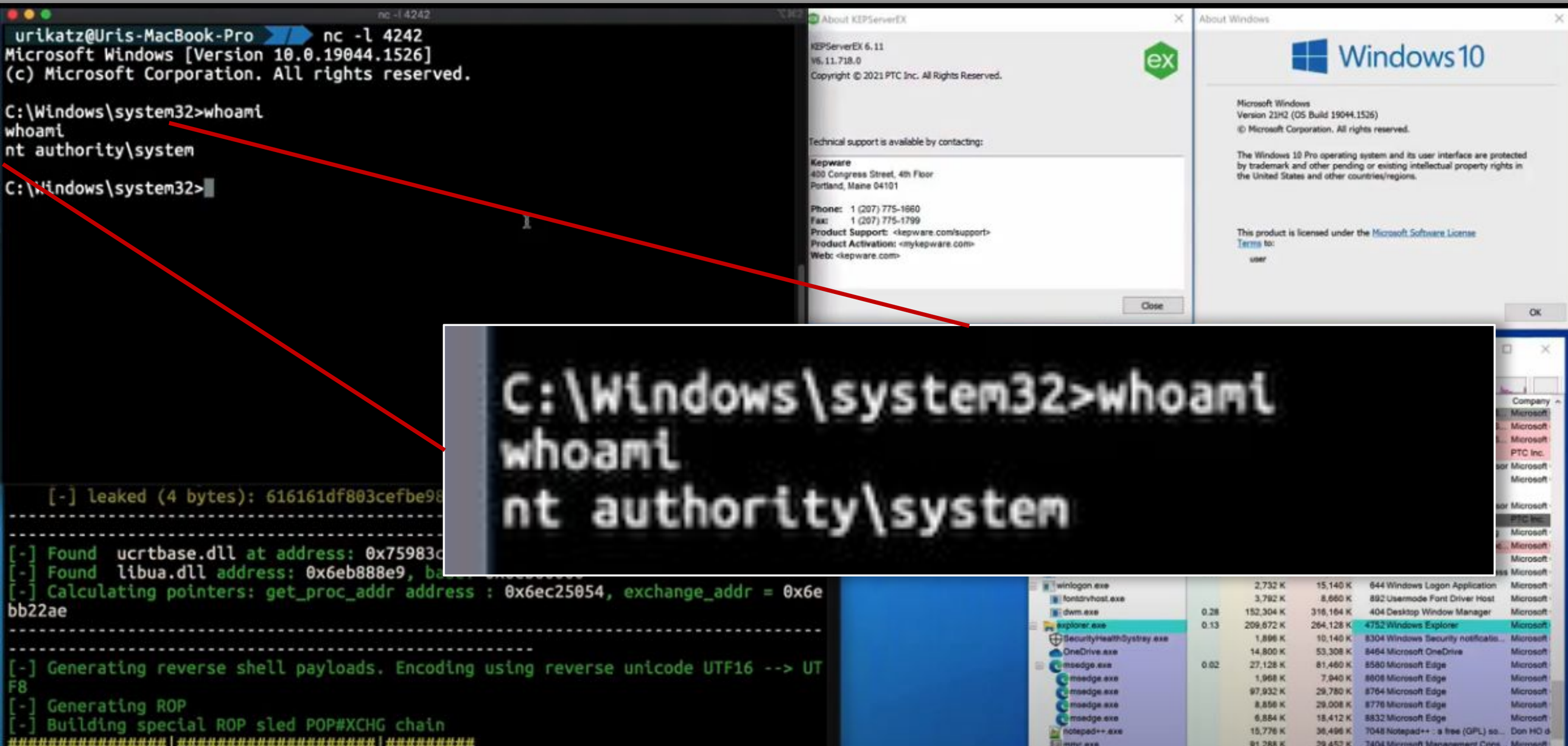


PTC Kepware RCE - Triggering



PTC Kepware RCE

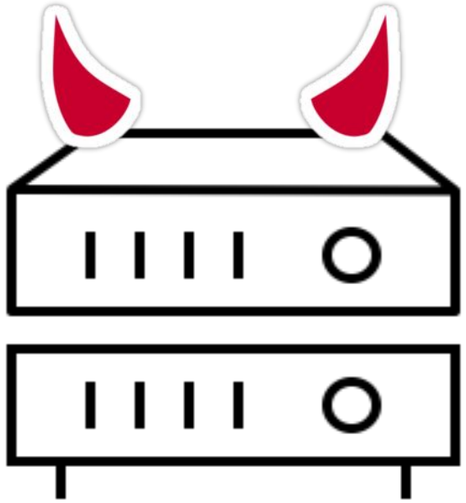
CVE-2022-2848
CVE-2022-2825



Vulnerabilities and Exploits

RCE - Clients

Attacking OPC-UA Clients



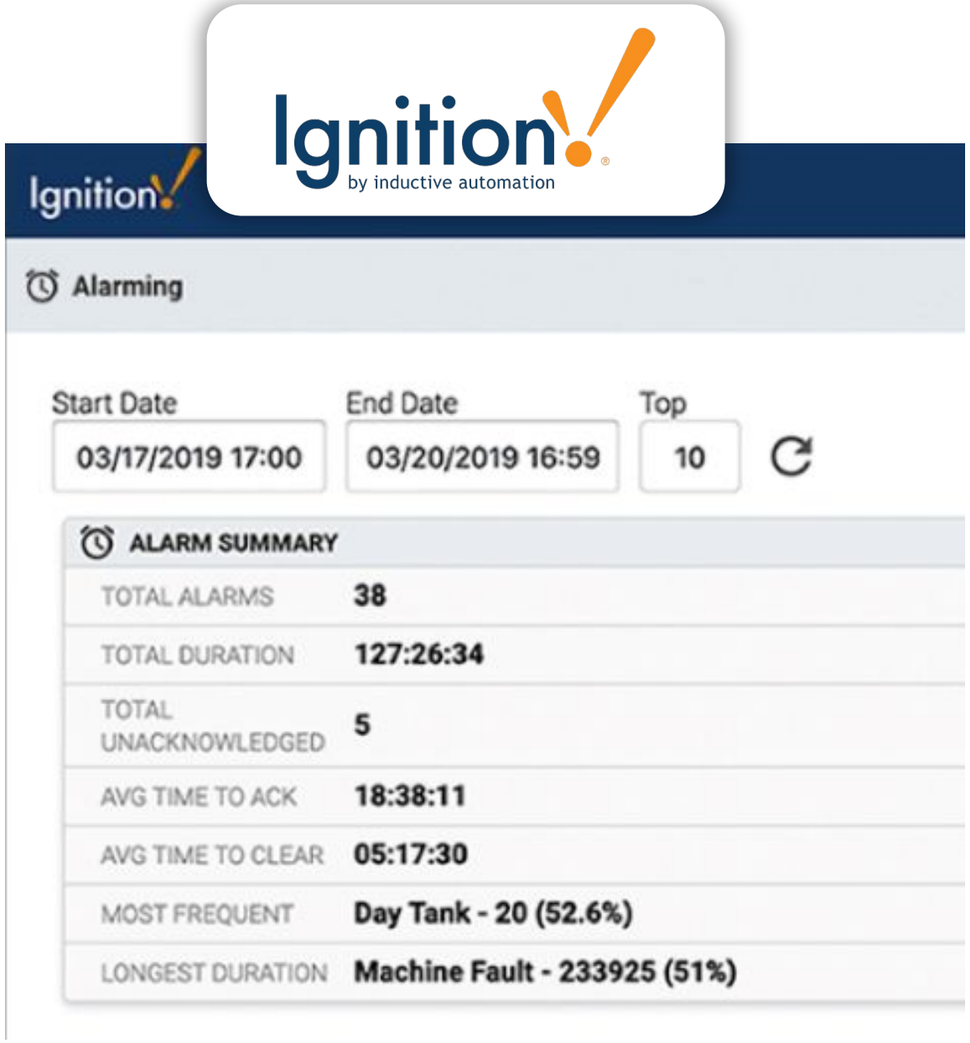
Malicious OPC-UA Server

OPC-UA

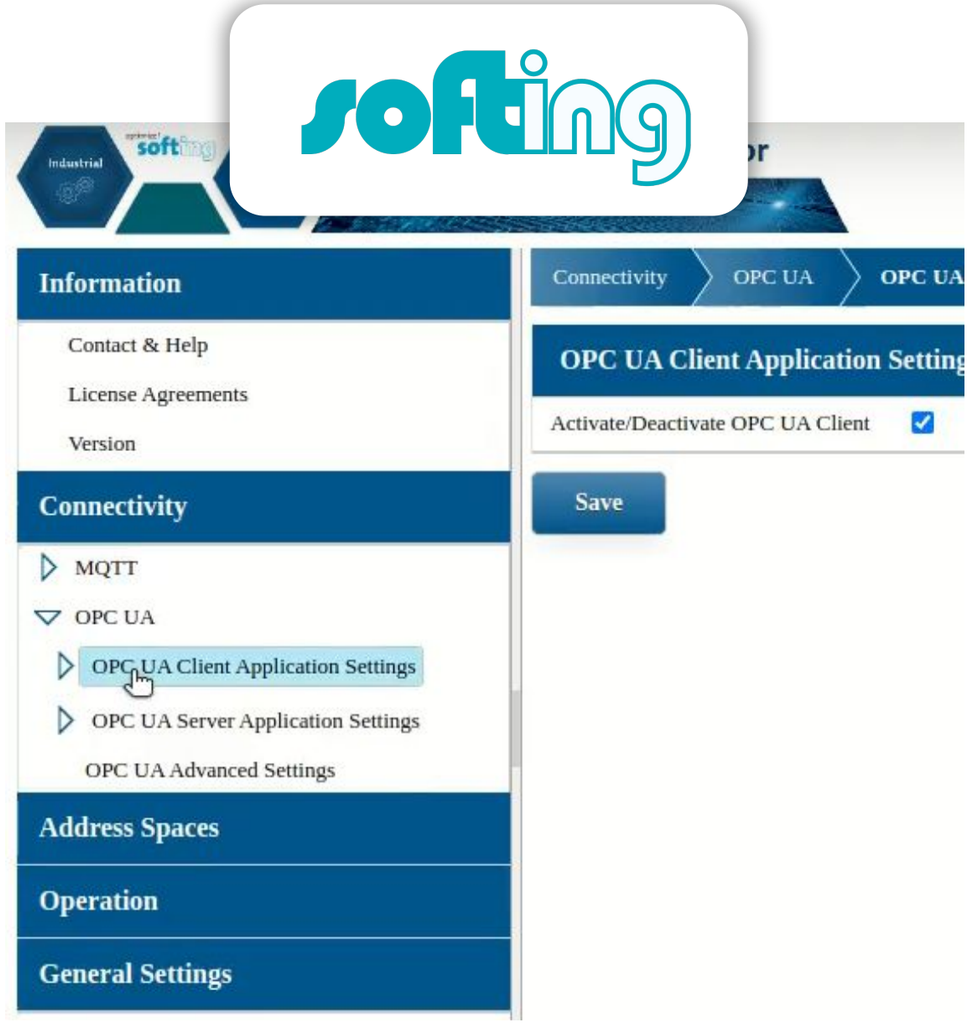


OPC-UA Client

Web-Based OPC-UA Clients

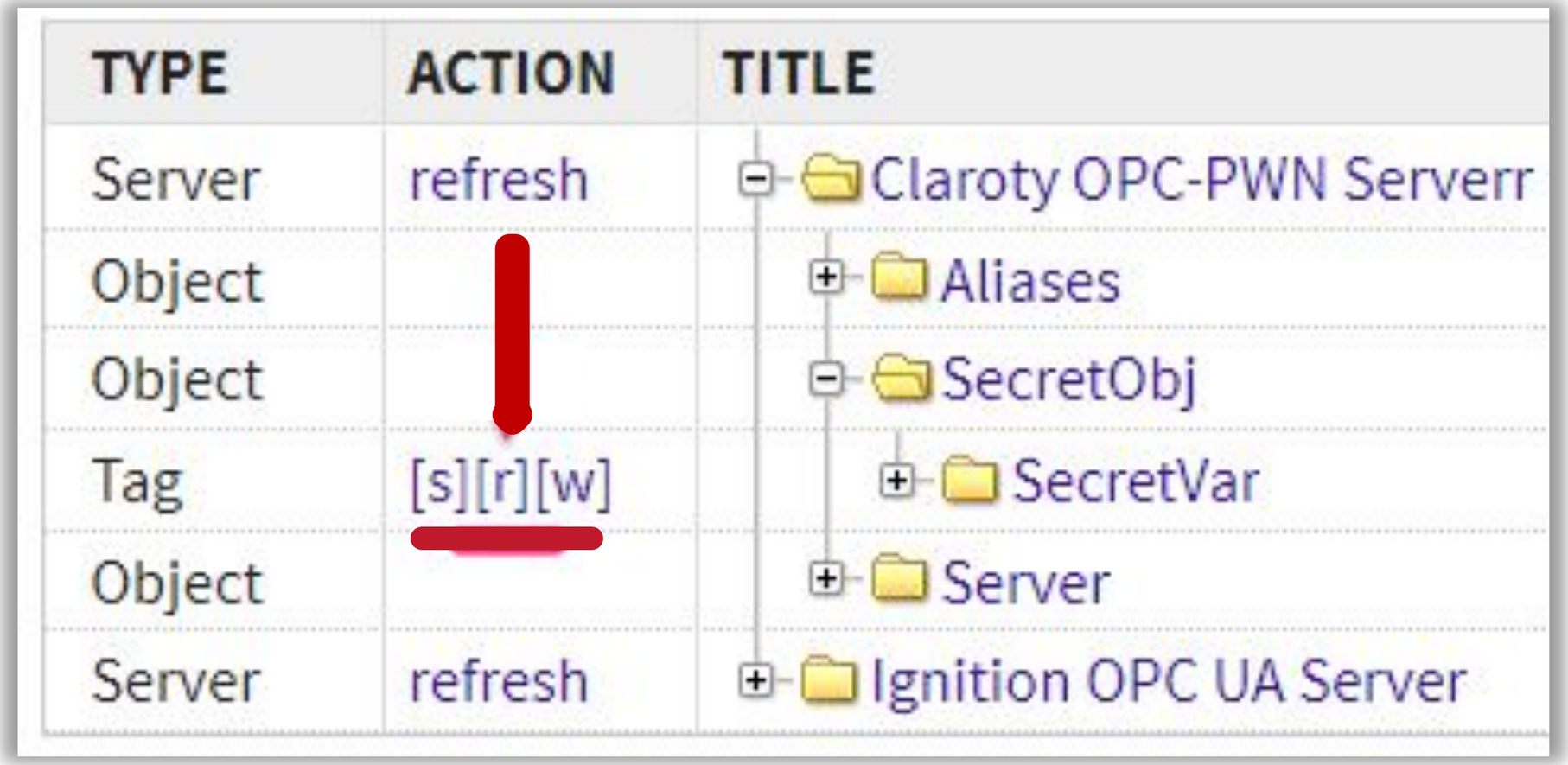


Inductive Automation
Ignition



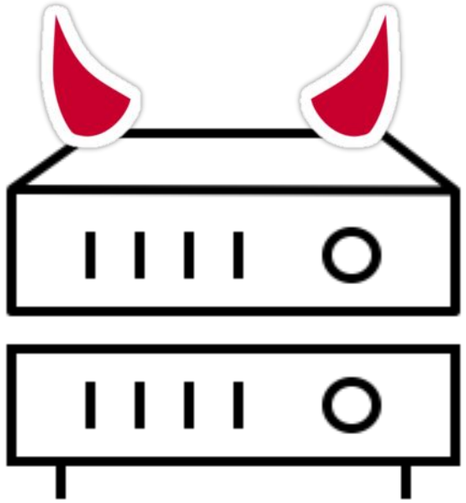
Softing
dataFEED edgeAggregator

XSS Over OPC-UA



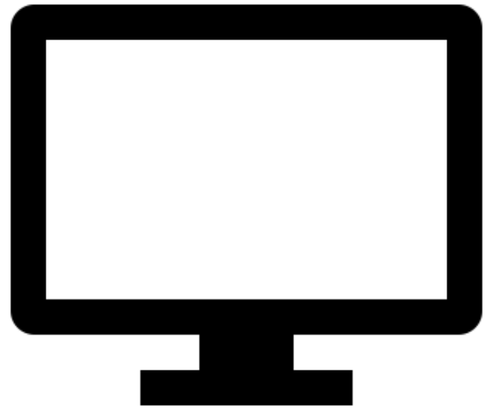
TYPE	ACTION	TITLE
Server	refresh	Claroty OPC-PWN Serverr
Object		Aliases
Object		SecretObj
Tag	[s][r][w]	SecretVar
Object		Server
Server	refresh	Ignition OPC UA Server

XSS Over OPC-UA



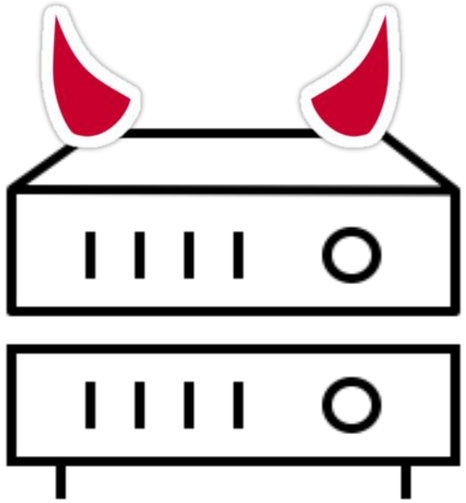
Malicious OPC-UA Server

OPC-UA



OPC-UA Client

XSS Over OPC-UA



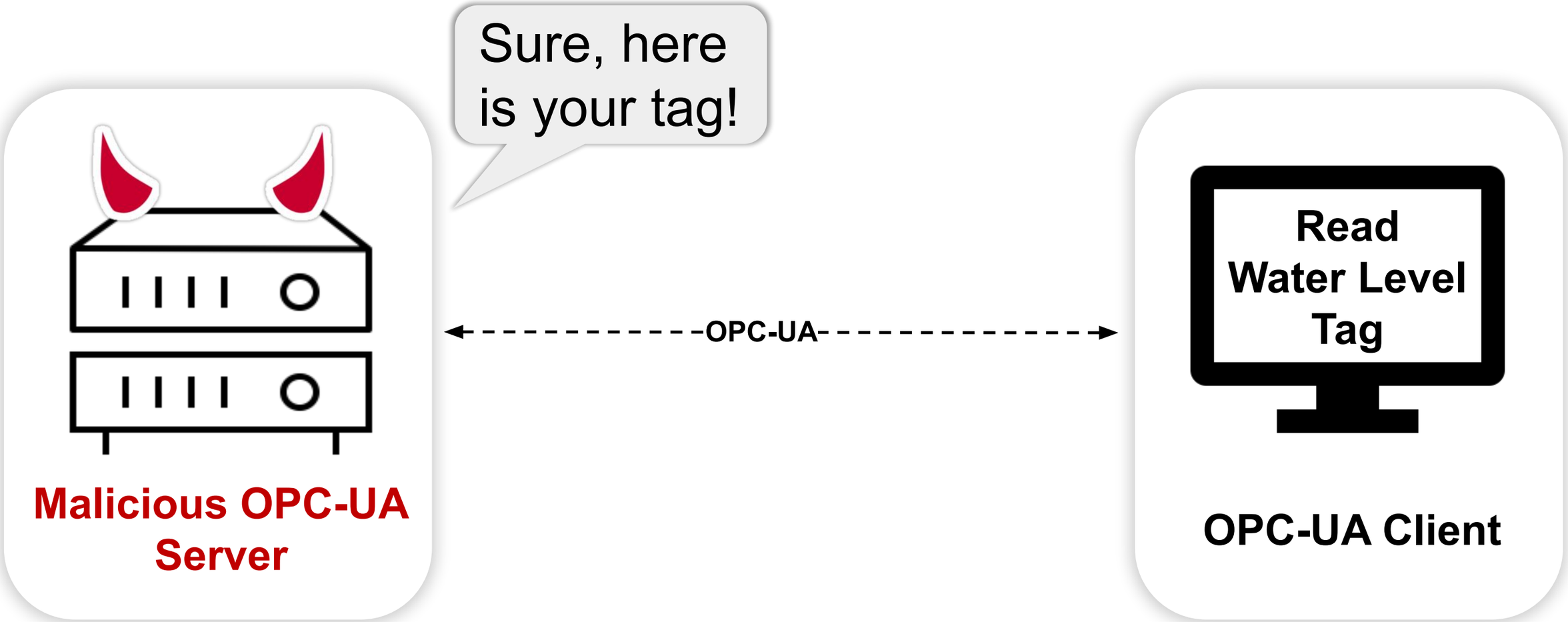
Malicious OPC-UA Server

OPC-UA

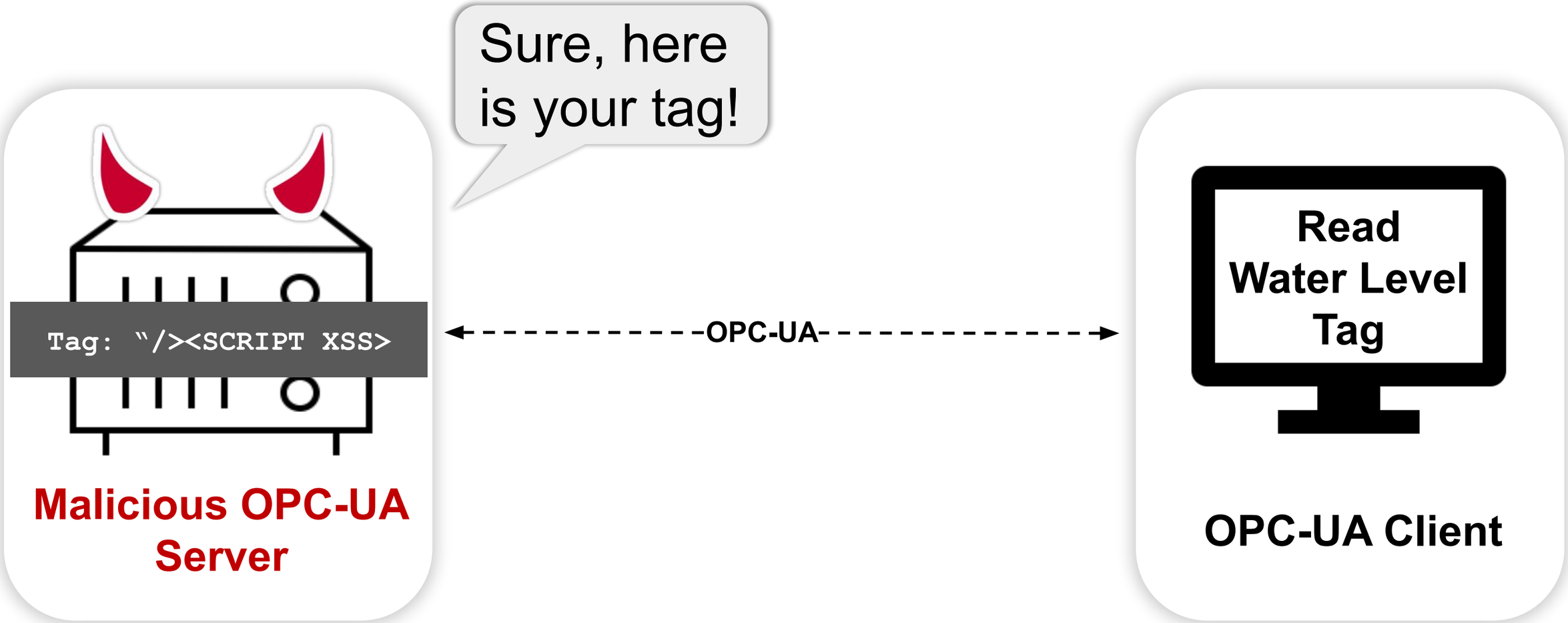


OPC-UA Client

XSS Over OPC-UA



XSS Over OPC-UA



XSS Over OPC-UA



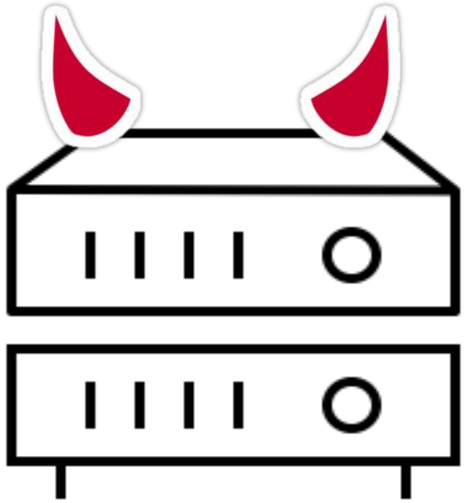
Malicious OPC-UA Server

OPC-UA



OPC-UA Client

XSS Over OPC-UA



Malicious OPC-UA Server

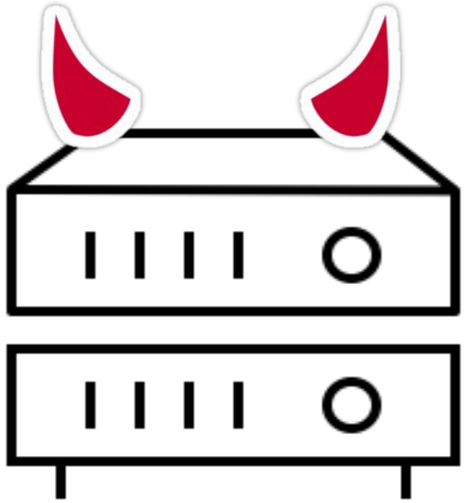
Tag: "/><SCRIPT XSS>

OPC-UA

Read Water Level Tag

OPC-UA Client

XSS Over OPC-UA



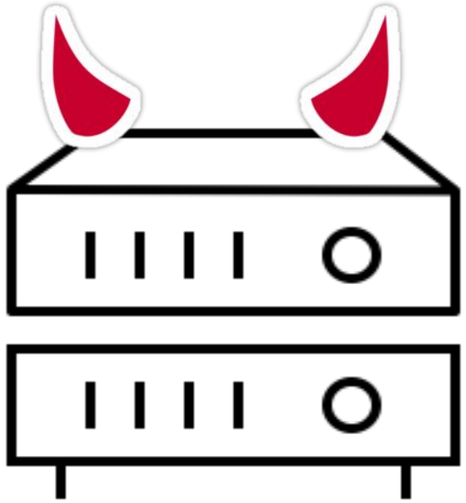
Malicious OPC-UA Server

OPC-UA



OPC-UA Client

XSS Over OPC-UA



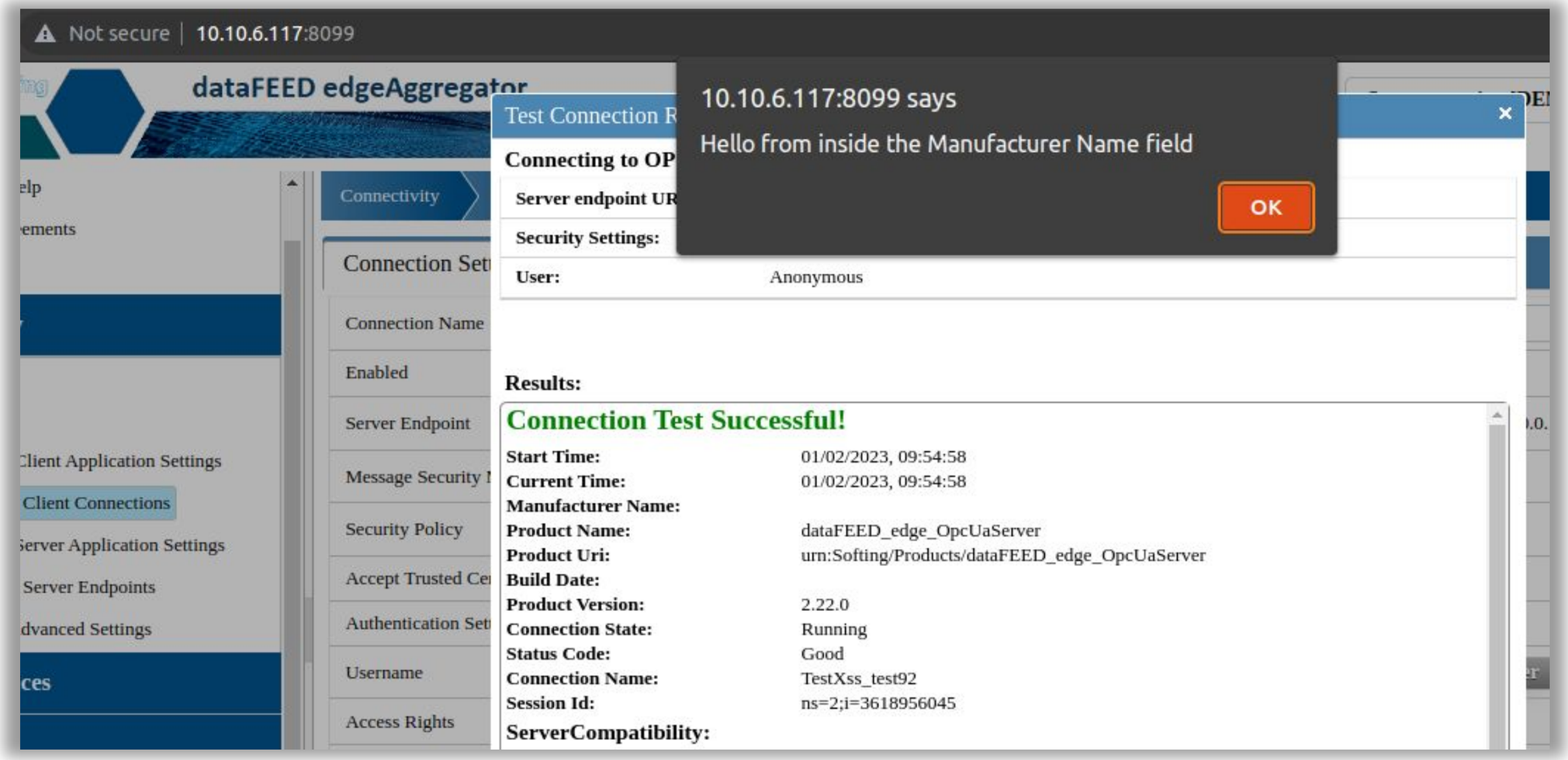
Malicious OPC-UA Server

OPC-UA

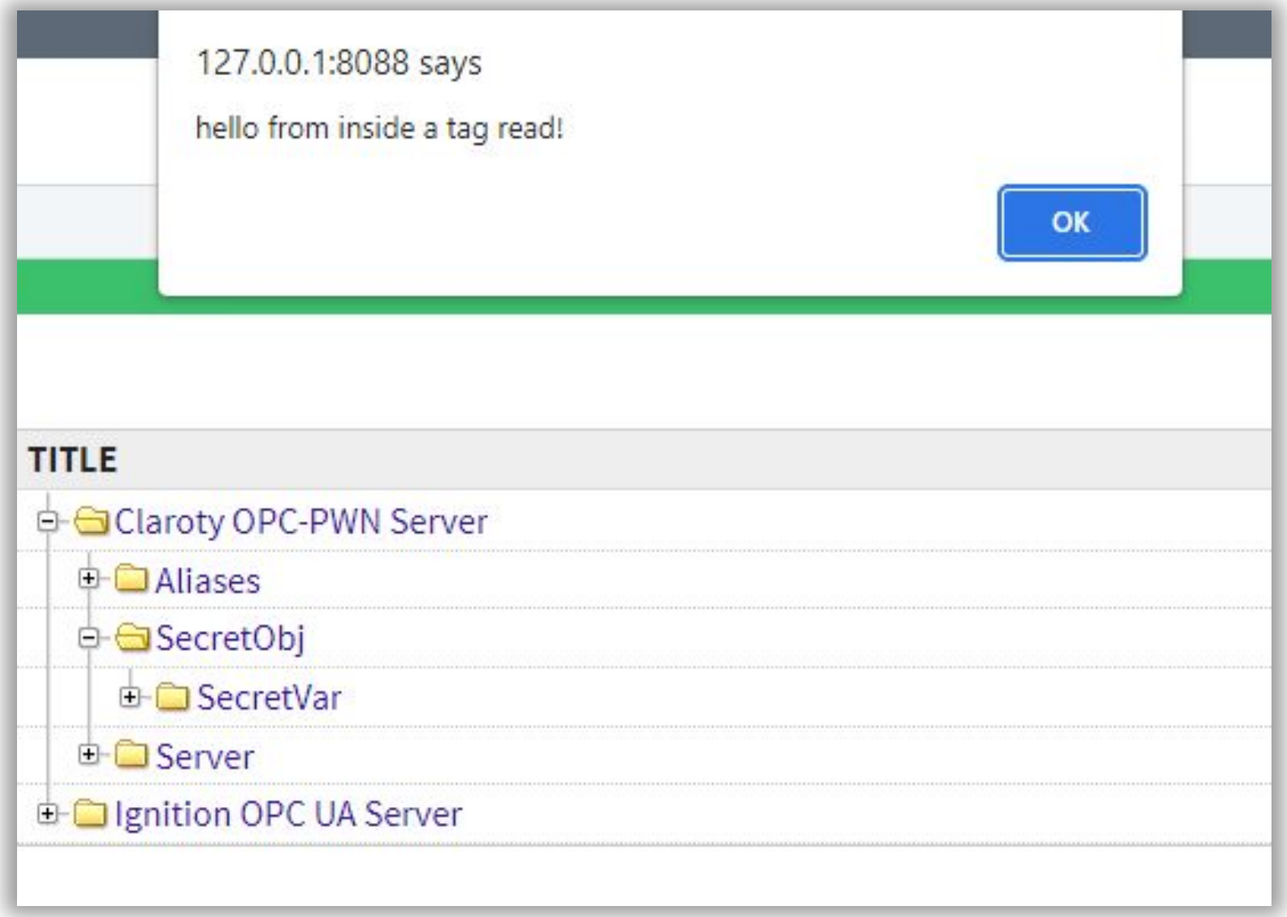


OPC-UA Client

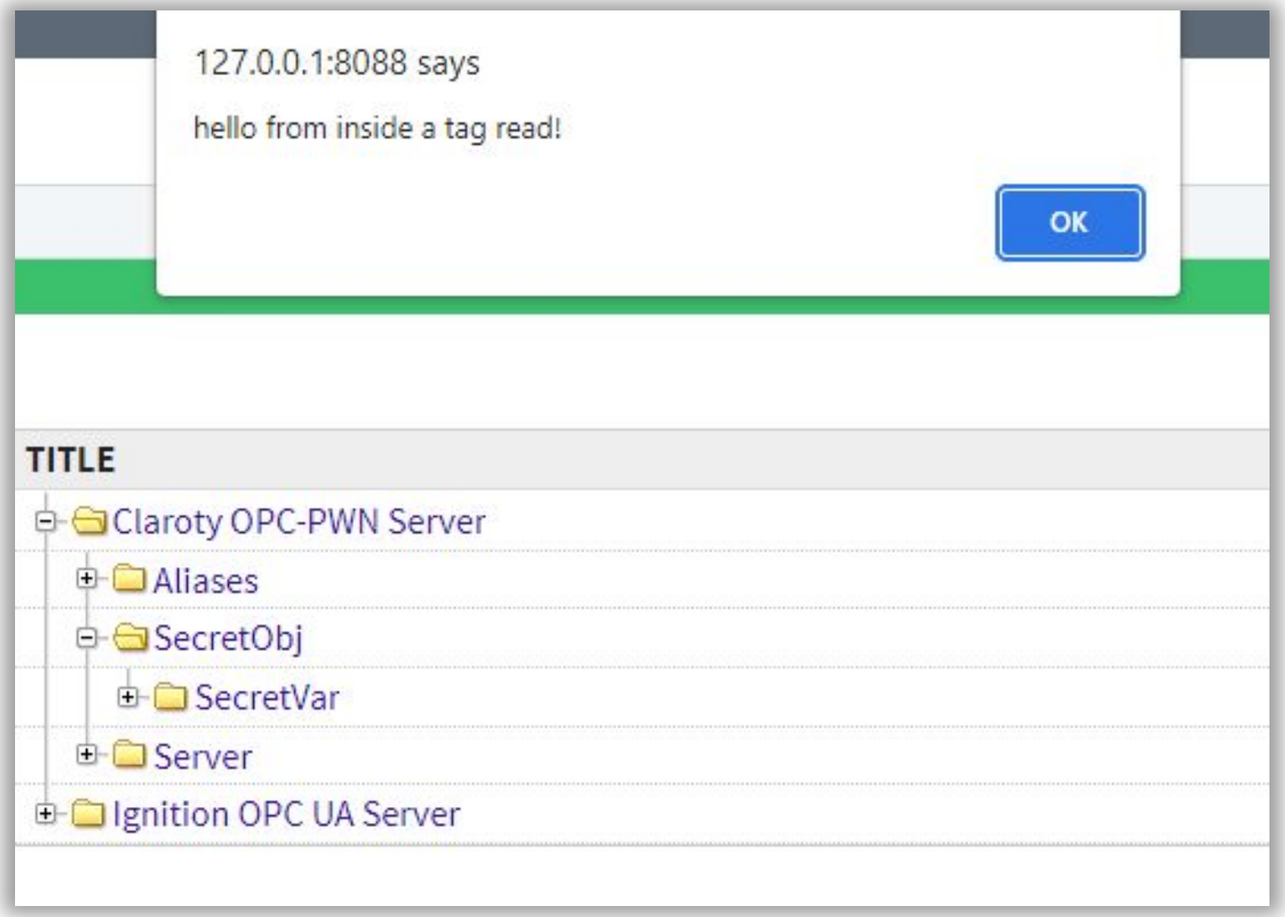
XSS Over OPC-UA

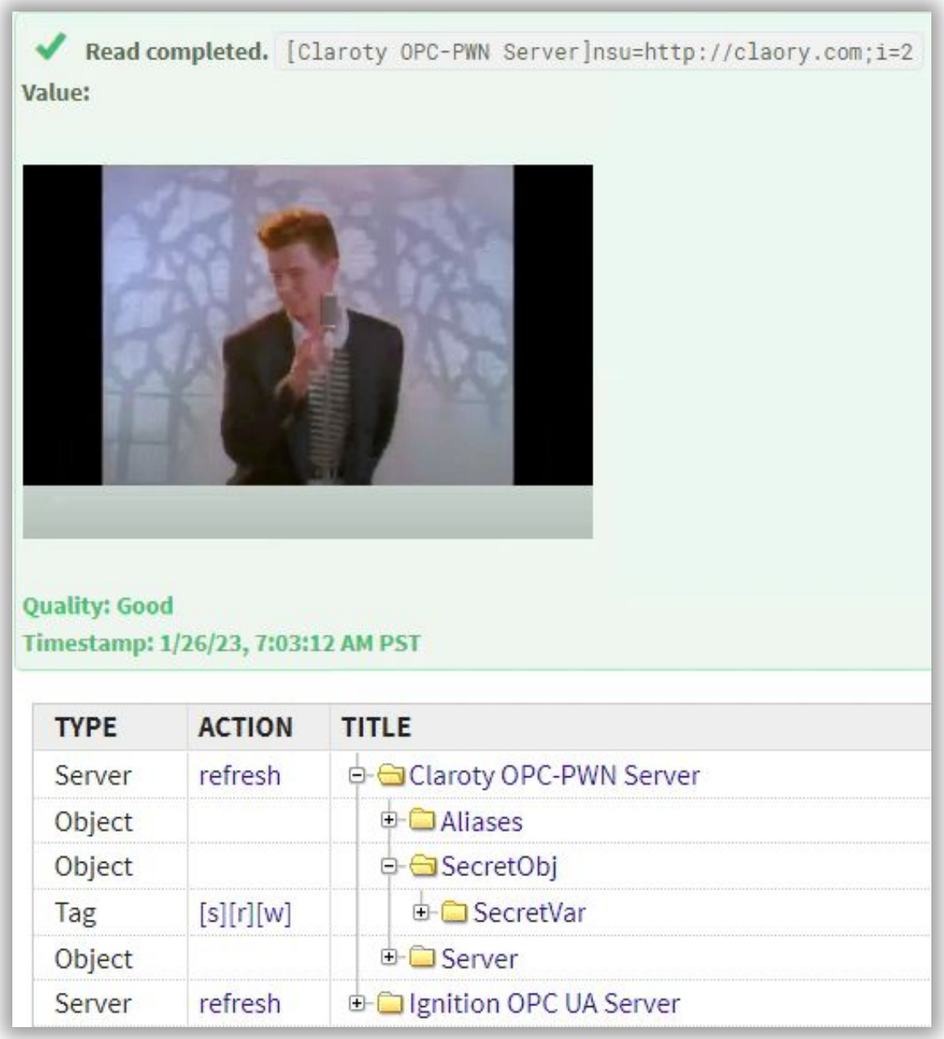


XSS Over OPC-UA



XSS Over OPC-UA





XSS Over OPC-UA to RCE

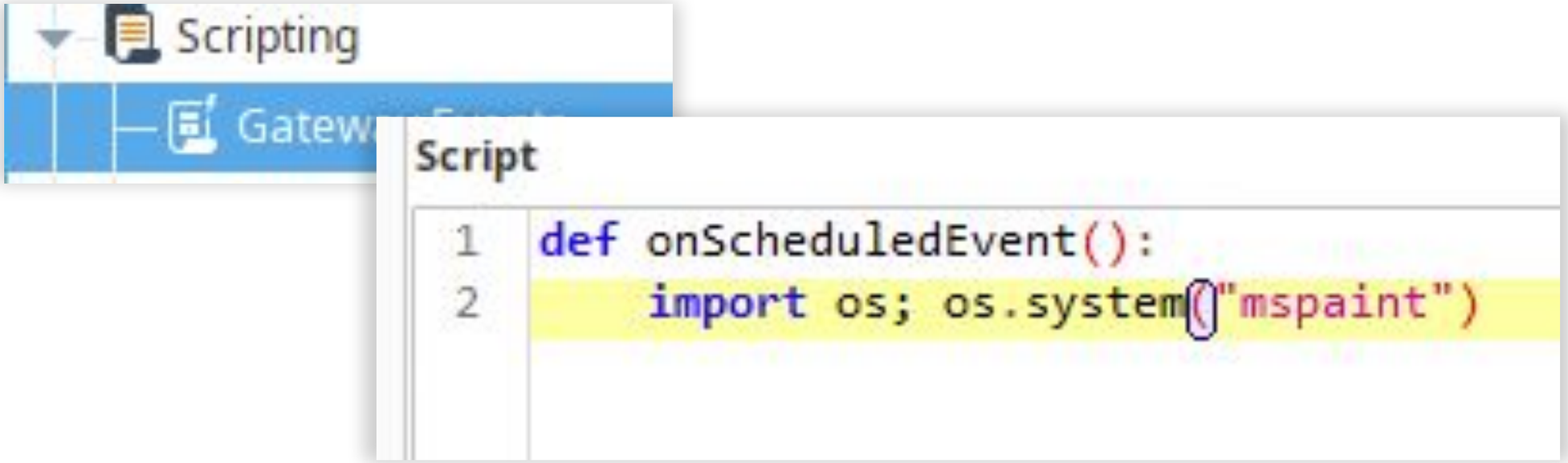
We are in the context of the OPC-UA client, how can we leverage into RCE?

Chain with more vulnerabilities

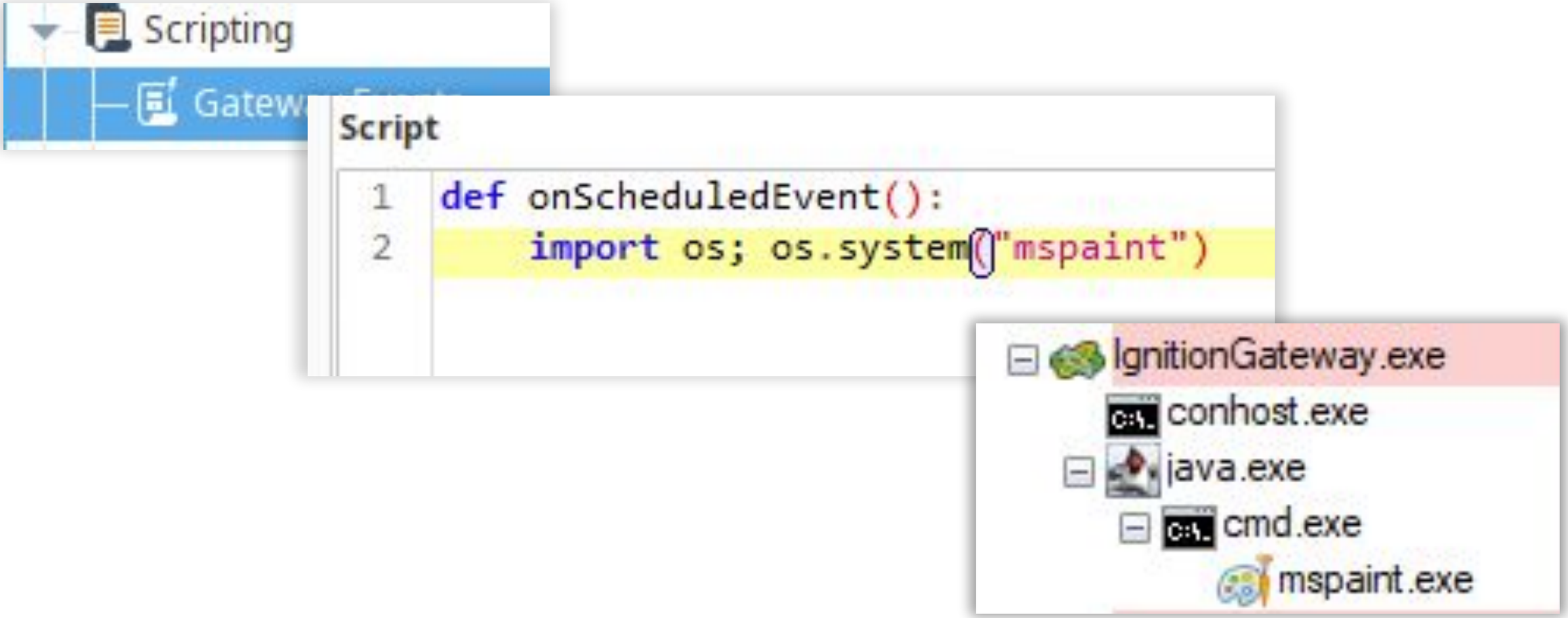
XSS Over OPC-UA to RCE



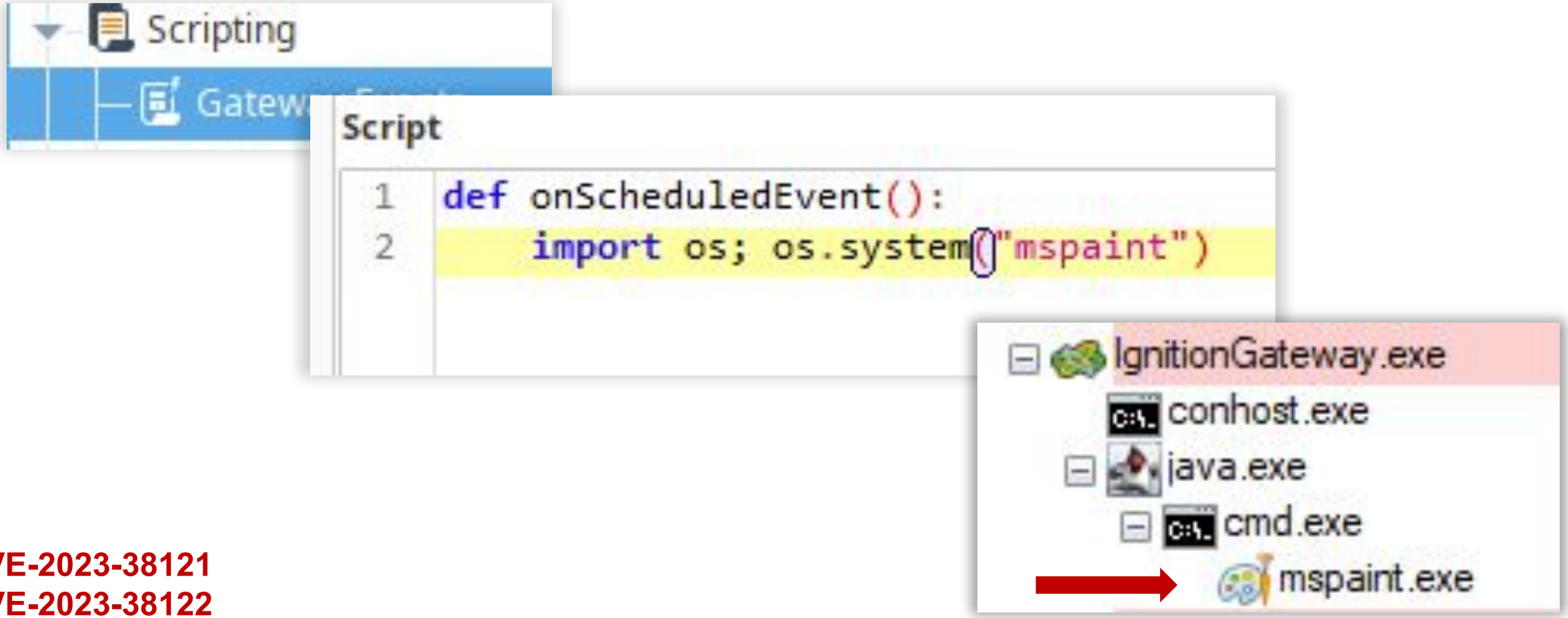
XSS Over OPC-UA to RCE



XSS Over OPC-UA to RCE

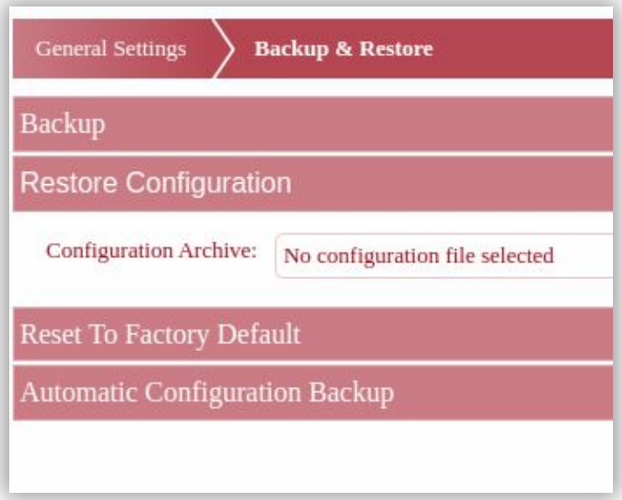


XSS Over OPC-UA to RCE



CVE-2023-38121
CVE-2023-38122
CVE-2023-38123
CVE-2023-38124

XSS Over OPC-UA to RCE





Backup

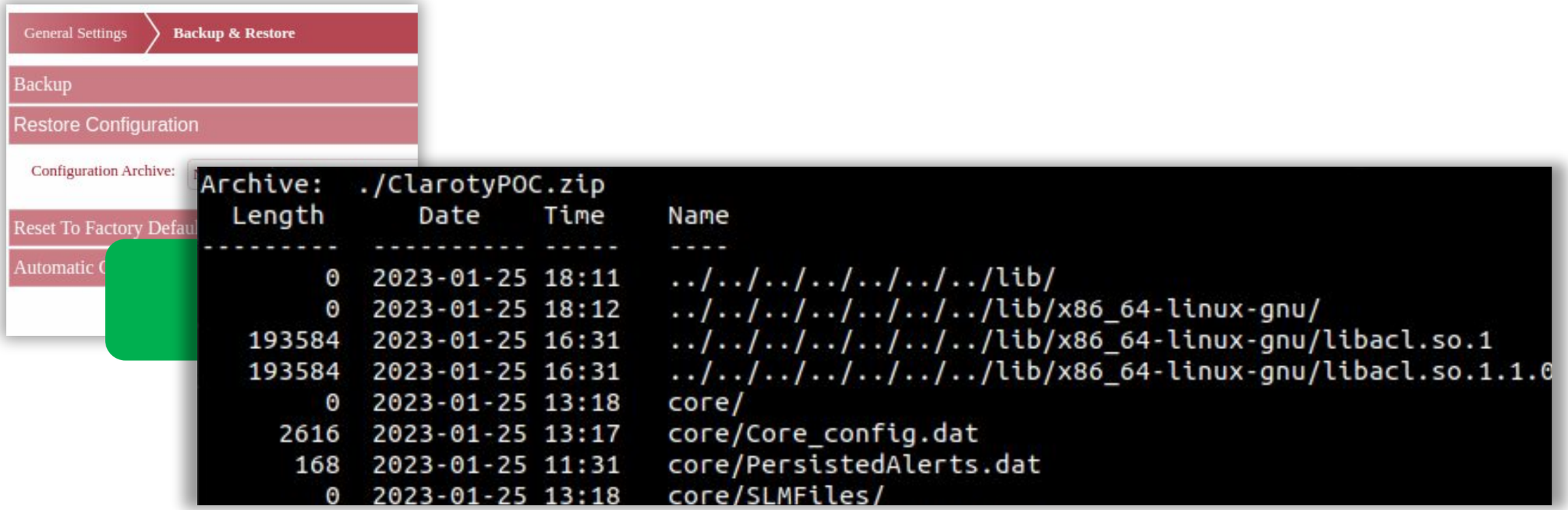
XSS Over OPC-UA to RCE



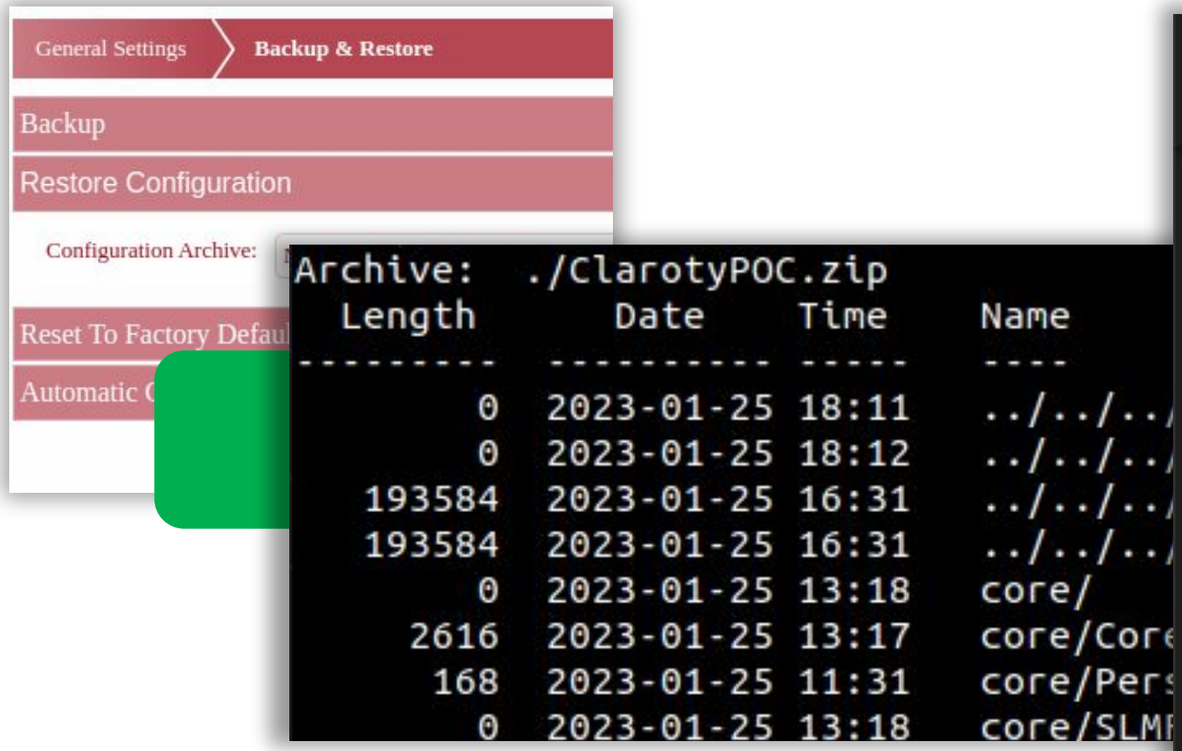


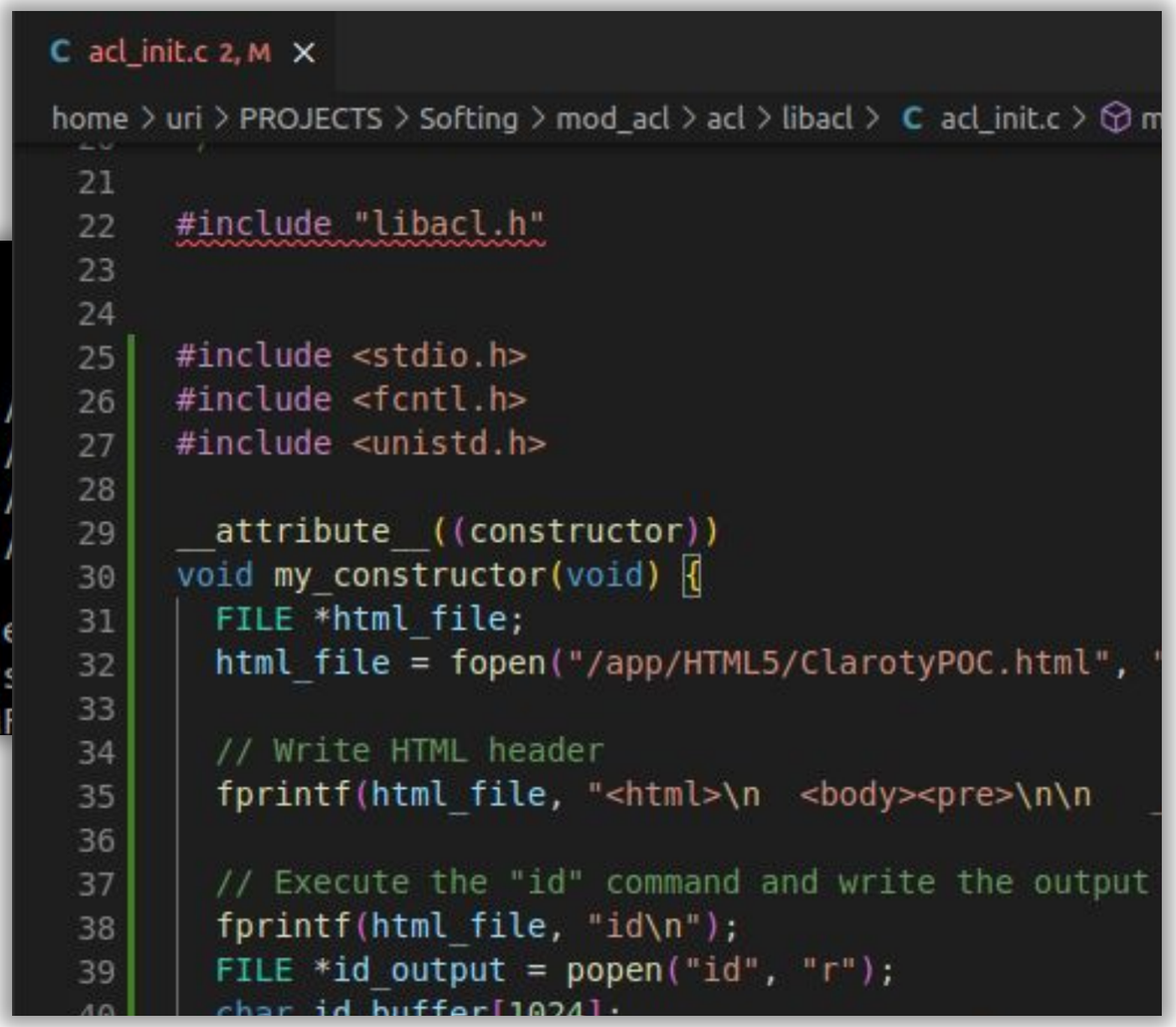
Backup

XSS Over OPC-UA to RCE



XSS Over OPC-UA to RCE





XSS Over OPC-UA to RCE



OPC-UA Exploitation Framework

Open-Source

Agenda

- What is OPC-UA?
- Protocol Stack Implementations
- Bits and Bytes
- Research Methodology
- Vulnerabilities and Exploits
- **OPC-UA Exploitation Framework**
- Summary

Results: 12 concepts, ~50 CVEs

Stack/Application Name	Lang	Complex Deep Nested Variants DoS	Worker Starvation DoS	Long Chunks DoS	Unlimited Monitored Items DoS	UTF8 - UTF16 Conversions
node-opcua	NodeJS	V	V	CVE-2022-21208	CVE-2022-24375	V
open62541	C	V	V	CVE-2022-25761	V	V
freeopcua (c++)	C++	V	V	V	CVE-2022-24298	V
python-opcua	Python	V	V	CVE-2022-25304	V	V
opcua-asyncio	Python	V	V	CVE-2022-25304	V	V
eclipse-milo	Java	V	V	V	CVE-2022-25897	V
ASNeG OpcUaStack	C++	V	V	CVE-2022-24381	V	V
locka99	Rust	CVE-2022-25903	V	CVE-2022-25888	V	V
Unified Automation	C++	V	V	V	Fixed, No CVE	V
OPC Foundation .NET Stack	C#	CVE-2021-27432 (*	V	CVE-2022-29864	V	V
Softing OPC UA SDK	C++	V	V	V	V	V
Prosys OPC UA	Java	V	CVE-2022-30551	V	V	V
OPC UA Legacy Java Stack	Java	V	CVE-2022-30551	V	V	V
Kepware KEPServerEX	C/C++	V	V	V	V	CVE-2022-2848 CVE-2022-2825

OPC-UA Exploit Framework

- Open source framework with all of our work
- Sharing after disclosed to all vendors + worked closely with them
- Based on our OPC-UA client
- Highly customizable with 12 out-of-the-box exploits



github.com/claroty/opcua-exploit-framework

Claroty OPC Exploit Framework

Attack Name	Description	Vulnerability Type	Function Keyword	CVE and Reference
Certificate Infinite Chain Loop	Some servers implemented the Certificate chain check by themselves and forgot to protect against a chain loop. Example: CertA is signed by CertB which is signed by CertA	Denial of Service	certificate_inf_chain_loop	CVE-2022-37013
Chunk Flooding	Sending huge amount of chunks without the Final chunk	Denial of Service	chunk_flood	CVE-2022-29864, CVE-2022-21208, CVE-2022-25761, CVE-2022-25304, CVE-2022-24381, CVE-2022-25888
Open Multiple Secure Channels	Flooding the server with many open channel requests leads to a denial of service	Denial of Service	open_multiple_secure_channels	CVE-2023-32787

Claroty OPC Exploit Framework

Function Call Null Dereference	Triggering an application crash after several OPC UA methods have been called and the OPC UA session is closed before the methods have been finished.	Denial of Service	function_call_null_deref	CVE-2022-1748
Malformed UTF8	Triggering an application crash after processing malformed UTF8 strings	Remote Code Execution	malformed_utf8	CVE-2022-2825, CVE-2022-2848
Race Change And Browse Address Space	Adding nodes to the server address space and removing the nodes in a loop while browsing the entire address space.	Denial of Service	race_change_and_browse_address_space	CVE-2023-32172
Unlimited Condition Refresh	Sending many ConditionRefresh method calls leads to uncontrolled memory allocations and eventually to a crash	Denial of Service	unlimited_condition_refresh	CVE-2023-27321

Claroty OPC Exploit Framework

Close Session With Old Timestamp	Sending bad timestamp on closing session leads to an uncaught stacktrace with sensitive information	Information Leakage	`close_session_with_old_timestamp`	CVE-2023-31048
Complex Nested Message	Sending a complex nested variant leads to a call stack overflow	Denial of Service / Information Leakage	`complex_nested_message`	CVE-2022-25903, CVE-2021-27432
Translate Browse Path Call Stack Overflow	Triggering a stack overflow exception in a server that doesn't limit TranslateBrowsePath resolving calls	Denial of Service	`translate_browse_path_call_stack`	CVE-2022-29866
Thread Pool Wait Starvation	Thread pool deadlock due to concurrent worker starvation	Denial of Service	`thread_pool_wait_starvation`	CVE-2022-30551
Unlimited Persistent Subscriptions	Flooding the server with many monitored items with 'delete' flag set to False leads to uncontrolled memory allocation and eventually to a denial of service	Denial of Service	`unlimited_persistent_subscriptions`	CVE-2022-25897,CVE-2022-24375,CVE-2022-24298

Agenda

- What is OPC-UA?
- Protocol Stack Implementations
- Bits and Bytes
- Research Methodology
- Vulnerabilities and Exploits
- OPC-UA Exploitation Framework
- Summary

Summary

Pwn2Own ICS: We participated and demonstrated our OPC-UA exploits in three Pwn2Own competitions - Pwn2Own ICS 2020, 2022, 2023

CVE: We found and reported on ~50 OPC-UA vulnerabilities/CVE across ~15 protocol stacks which affects hundreds of OPC-UA products.

Exploit Technique: We developed ~12 unique exploit techniques that are universal and affected multiple vendors and pushed to change the specs.

Open-Source Tools: We released two OOS tools including OPC-UA network fuzzer and OPC-UA exploitation framework.

OPC-UA Specifications: we helped to improve the specifications and pushed the vendors towards better and more secure products.