Deep Dive Into PIPEDREAM's OPC UA Module, MOUSEHOLE

By Sam Hanson

Published: 2023-05-05 · Archived: 2026-04-02 10:50:37 UTC

In April of 2022, Dragos published a whitepaper and hosted a webinar discussing PIPEDREAM, the seventh known industrial control systems (ICS)-specific malware developed by the CHERNOVITE threat group. Dragos followed up with a blog titled, "Analyzing PIPEDREAM: Results from Runtime Testing." Continuing this research, Dragos is releasing additional information on the Open Platform Communications Unified Architecture (OPC UA) module nicknamed MOUSEHOLE, focusing on further analysis and runtime testing results.

In the first post of this two-part blog, Dragos analysts briefly provide background on OPC UA fundamentals, then dive into a high-level overview of MOUSEHOLE's capabilities, discuss the open-source Python library utilized by MOUSEHOLE, and finally highlight other libraries that an adversary could abuse in a similar way. In part two, Dragos analysts discuss the static and dynamic analysis of MOUSEHOLE, an experiment we conducted to showcase MOUSEHOLE's capabilities, and what industrial control system (ICS) asset owners and security practitioners can do to protect against rogue OPC UA clients.

What is OPC UA?

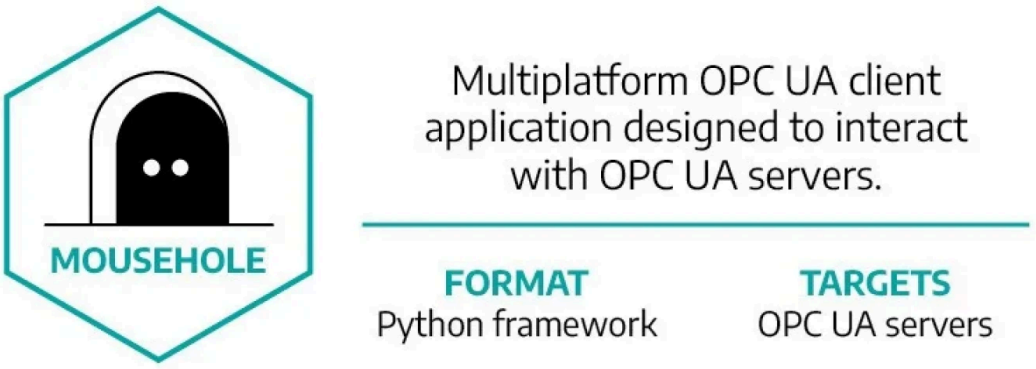
Open Platform Communications Unified Architecture (OPC UA) is a popular industrial protocol allowing for data communication between various devices and systems. OPC UA was created to better address the needs of the growing industrial automation market, moving away from its predecessor, OPC Classic's reliance on Windows and COM/DCOM technology. OPC UA is platform independent, meaning it can be used on Windows, Linux, or MacOS hosts, and includes all functionalities found in the OPC Classic specification.1 OPC UA was released in 2008 by the OPC Foundation and later added as an IEC standard (IEC 62541).

Simply put, OPC UA is an industrial and Internet of things (IoT) communication standard and can directly impact how critical systems function. For example, logic running a programmable logic controller (PLC) could use variables set and modified by a separate device through the OPC UA protocol.

It is important to remember that while MOUSEHOLE abuses the OPC UA protocol, there's nothing inherently insecure about OPC UA. In fact, the protocol provides a variety of impressive security settings and configurations.2, 3, 4 While vendors who produce OPC UA server software may not require strong security settings to be configured by the user,5 that is a vendor implementation issue and *not* an OPC UA problem.

For more information on the technical details of the OPC UA specification, please see the OPC Foundation's online reference.

What Is MOUSEHOLE?



MOUSEHOLE is one of five modules in PIPEDREAM, the seventh known industrial control systems (ICS)-specific malware. MOUSEHOLE is a Python program that functions as an OPC UA client application. It is designed for easy interaction with OPC UA servers from the command line and contains various capabilities, including:

- Scanning a network for an OPC UA server
- Brute forcing the authentication mechanism
- Reading the structure of a server
- Reading and writing to specific node attributes
- Setting various security settings such as security mode, policies, certificates, and private keys

An adversary with an understanding of a victim's operational technology (OT) environment could modify a node's value attribute on a poorly secured OPC UA server, causing a direct impact on operations, including the possibility of Loss of Control to connected control systems.

MOUSEHOLE leverages the open-source Python library, *python-opcua*, which significantly reduces the complexity of interacting directly with the protocol, thus lowering the bar of sophistication required to impact operations successfully.

MOUSEHOLE and the Python-OPCUA Library

The *python-opcua* library, while deprecated, is available on GitHub for anyone to download and use. This API makes it incredibly easy for a programmer to connect, authenticate, and send requests to an OPC UA server with only a few API calls. The library exposes various services to the programmer, such as the read or write attribute service, which can be called upon by the client and executed by the server. The library achieves this by implementing internal Python classes and objects that comply with the OPC UA Service Set protocol specification. These Python objects are sent in binary format, interpreted, and executed by the server.

Let's walk through an example client application to demonstrate how simple it is to connect to and manipulate an OPC UA server. We will discuss the *get_value* function (which reads a node's value attribute) to demonstrate how the API works under the hood. Our script will connect to a poorly secured server (anonymous authentication enabled), read a node value attribute, and finally, write a value to the node. In total, the final script looks like the following:

```python
from opcua import *

# First, generate the client variable...
client = Client('opc.tcp://192.168.97.2:4840')

# Connect to the server...
client.connect()

# Read a node's value attribute...
node = client.get_node('ns=2;s=Channel1.HighPressureSetpoint')
node_value = node.get_value()

# Write a node's value attribute...
node.set_value(100)
```

Only six lines of code are needed to connect, receive, and send data to the server, which are the foundational components of MOUSEHOLE's functionality. However, a lot is happening under the hood that the programmer may be blissfully unaware of.

For example, the *get_value* method executes a series of functions that populate a Python *ReadRequest* object representing the OPC UA read service. This *ReadRequest* object is binarized and sent to the server, which is then interpreted and executed, as shown in Figure 2. The value is sent back to the client and, in our example script, stored in the *node_value* variable seen in Figure 1.

```python
def read(self, parameters):
    self.logger.info("read")

    # Instantiate and send ReadRequest object
    request = ua.ReadRequest()
    request.Parameters = parameters
    data = self._uasocket.send_request(request)

    # Obtain response in binary format, convert to ReadResponse object.
    response = struct_from_binary(ua.ReadResponse, data)
    self.logger.debug(response)
    response.ResponseHeader.ServiceResult.check()
```

The *set_value* method works similarly but instead sends a *WriteRequest* object representing the write service to the server. The API makes it as simple as possible to interact with the OPC UA server without exposing the programmer to the complexities of OPC UA or its internal structures.

Open-Source Industrial Protocol Libraries

From a programming perspective, MOUSEHOLE is not a sophisticated tool. Most of the code is a simple command line interface for the *python-opcua* library, where much of the complexity is hidden. The programmer

must understand only the highest-level function calls and objects. This abstraction of knowledge is a double-edged sword; it simplifies the job of a legitimate developer while allowing adversaries to quickly develop programs that can cause serious industrial impact with little required expertise in the technology and protocols.

There are dozens of open-source industrial protocol libraries and APIs on GitHub that could be abused in a similar fashion, including Modbus, BACnet, DNP3, IEC 104, IEC 61850 Ethernet/IP and CIP, Ethercat, and many more.6 Some of these libraries and APIs are incredibly advanced and provide significant capabilities to the user. For example, the open-source PyModbus implementation is a full-featured server and client application with a command line interface. An adversary could download this tool and have a significantly more capable Modbus equivalent to MOUSEHOLE. Dragos has no evidence that these libraries have been leveraged to create malicious tools. Nonetheless, these libraries could be abused in the future.

In Summary

CHERNOVITE's creation of a malicious OPC UA tool is more indicative of OPC UA's ubiquity than the protocol's security. CHERNOVITE could have easily used any number of open-source libraries that implement common industrial protocols, many of which are less secure than OPC UA. As the proliferation of ICS tools and knowledge expands, it is paramount that defenders understand what emerging threats may exist and take action to mitigate the risk.

Part two of our blog covers the runtime experiments we conducted with MOUSEHOLE and best practices for OPC UA server security. Stay tuned! In the meantime, be sure to check out other PIPEDREAM-related content on our blog:

- CHERNOVITE's PIPEDREAM Malware Targeting Industrial Control Systems (ICS)
- Responding to CHERNOVITE's PIPEDREAM with Dragos Global Services
- Detecting CHERNOVITE's PIPEDREAM with the Dragos Platform

Get the Complete Analysis

Learn more about the discovery and capabilities of CHERNOVITE's PIPEDREAM malware in our whitepaper.

DOWNLOAD WHITEPAPER

Source:

1. Unified Architecture – opcfoundation.org
2. Exploring Cybersecurity and OPC UA for a Secure Systems Architecture – opcconnect.opcfoundation.org
3. OPC 10000-2: Security – reference.opcfoundation.org
4. OPC UA Security Analysis – opcfoundation.org
5. Security Analysis of Vendor Implementations of the OPC UA Protocol for Industrial Control Systems – arxiv.org
6. ICS-Security-Tools – github.com

Sam Hanson is a Vulnerability Analyst on the Intelligence Research Team. Sam graduated from the University of Minnesota – Twin Cities in 2020 with a Computer Science degree and a focus on Computer Security.
