Coyote in the Wild: First-Ever Malware That Abuses UI Automation

akamai.com/blog/security-research/active-exploitation-coyote-malware-first-ui-automation-abuse-in-the-wild

Written by

Tomer Peled

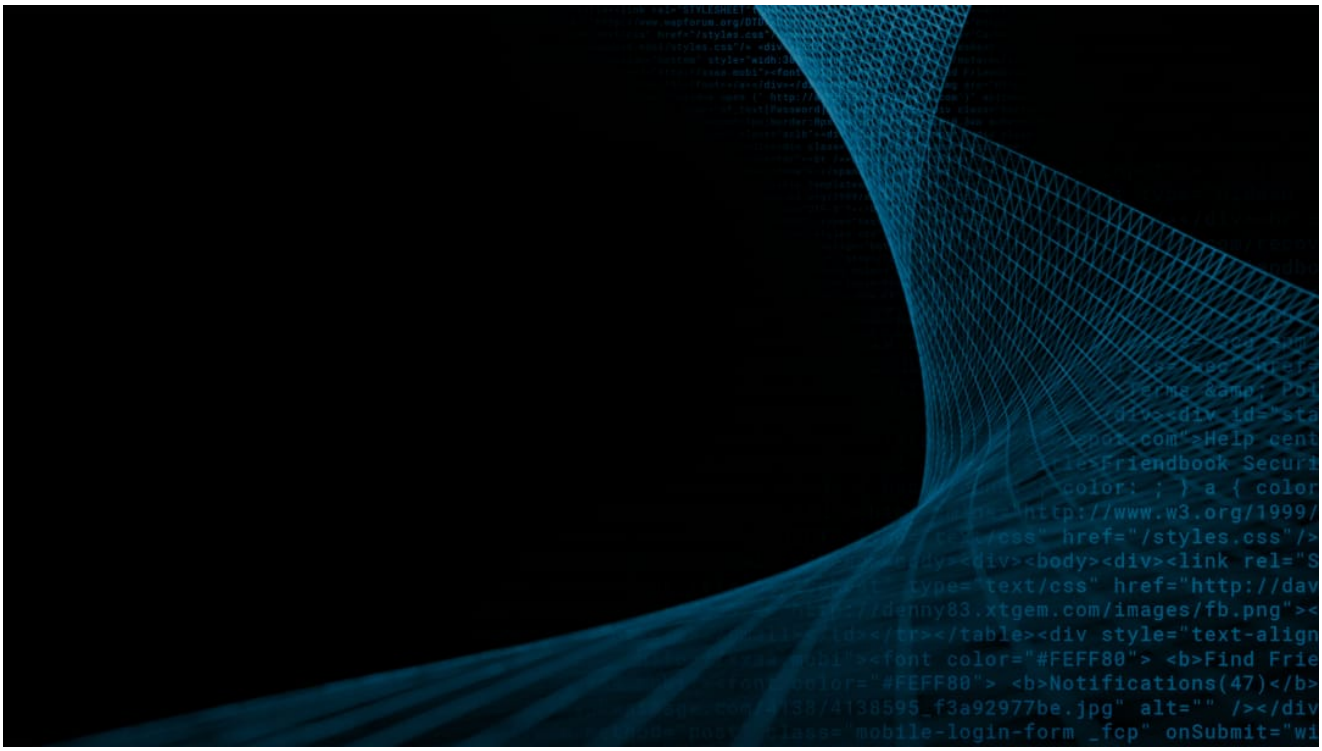
July 22, 2025

Written by

Tomer Peled

Tomer Peled is a Security Researcher at Akamai. In his daily job, he conducts research ranging from vulnerability research to OS internals. In his free time, he likes to cook, do Krav Maga, and game on his PC.

Share



This UIA abuse is the latest of these malicious Coyote tracks in their digital habitat since its discovery in February 2024.

Executive summary

- Akamai researchers previously outlined the potential for malicious use of UIA.
- Now, Akamai researchers have analyzed a new variant of the Coyote malware that is the first confirmed case of maliciously **using Microsoft's UI Automation (UIA) framework in the wild**.

- The new Coyote variant is targeting Brazilian users, and uses UIA to extract credentials linked to 75 banking institutes' web addresses and cryptocurrency exchanges.
- To help prevent Coyote infections and UIA abuse more broadly, we've included indicators of compromise and additional detection measures in this blog post.

Jump to detections

Introduction

In December 2024, we published a blog post that highlighted how attackers could abuse Microsoft's UIA framework to steal credentials, execute code, and more. Exploitation was only a proof of concept (PoC) — until now.

Approximately two months after the publication of that blog post, our concerns were validated when a variant of the banking trojan malware **Coyote** was observed abusing UIA in the wild — marking the first known case of such exploitation.

This UIA abuse is the latest of these malicious Coyote tracks in their digital habitat since its discovery in February 2024.

In this blog post, we take a closer look at the variant to better understand how UIA is being leveraged for malicious purposes, and what it means for defenders.

What is Coyote malware?

Coyote is a well-known malware family that was discovered in February 2024 and has caused significant damage in the Latin America region ever since. Coyote is a trojan malware that employs various malicious techniques, such as keylogging and phishing overlays, to steal banking information.

It uses the Squirrel installer to propagate (hence the name "Coyote," which pays homage to the coyotes' nature to hunt squirrels). In one of its most well-known campaigns, Coyote targeted Brazilian companies in an attempt to deploy an information stealing Remote Access Trojan within their systems.

After the initial discovery of Coyote, many security researchers uncovered details of its operations and provided in-depth technical analyses. One such examination, published by Fortinet in January 2025, shed light on Coyote's internal workings and attack chain.

UIA abuse

We've expanded on those analyses and discovered one new key detail: Coyote now leverages UIA as part of its operation. Like any other banking trojan, Coyote is hunting banking information, but **what sets Coyote apart is the way it obtains this information, which involves the (ab)use of UIA**.

Coyote gets rabid

During its infection process, Coyote sends the command and control server detailed information about each victim. This includes the computer name, user name, and various other system attributes. However, the most notable pieces of information — the one that Coyote invests significant effort to obtain — are the financial services used by the victim.

Initially, the malware will use a classic, very common approach: Coyote invokes the *GetForegroundWindow()* Windows API to obtain a handle to the currently active window. Once it retrieves the window handle, the malware will compare the window title to a list of hardcoded web addresses belonging to targeted banks and crypto exchanges.

The interesting UIA logic kicks in when the title doesn't match any of the addresses Coyote is looking for. **If no match is found Coyote will then use UIA to parse through the UI child elements of the window in an attempt to identify browser tabs or address bars**. The content of these UI elements will then be cross-referenced with the same list of addresses from the first comparison.

To do this, Coyote creates the *UIAutomation* COM object with the foreground window as its top element (Figure 1).

```
CUIAutomation cuiautomation = (CUIAutomation)Activator.CreateInstance(
Marshal.GetTypeFromCLSID(
new Guid("FF48DBA4-60EF-4201-AA87-54103EEF594E"))
);
IUIAutomationElement iuiautomationElement=cuiautomation.ElementFromHandle(ForgroundWindow);
```

Fig. 1: UIA creation

Coyote will then iterate through each sub-element of the foreground application to find the web address of a tab (Figure 2).

```
for (int i = 0; i < iuiautomationElementArray.Length; i++)
{
    try
    {
        IUIAutomationElement element = iuiautomationElementArray.GetElement(i);
        if (Malware.<>o__127.<>p__0 == null)
        {
            Malware.<>o__127.<>p__0 = CallSite<Func<CallSite, object, IUIAutomationValuePa
            IUIAutomationValuePattern), typeof(Malware)));
        }
        IUIAutomationValuePattern iuiautomationValuePattern = Malware.<>o__127.<>p__0.Tarc
        ));
        if (iuiautomationValuePattern != null && !string.IsNullOrEmpty(iuiautomationValuel
        {
            text = iuiautomationValuePattern.CurrentValue;
            if (Regex.IsMatch(text,  ^(https:\/\/)?[a-zA-Z0-9\-\.]+(\.[a-zA-Z]{2,4}).*$
                if (!text.StartsWith(
```

Fig. 2: UIA iterates through sub-elements

Once the web address has been found, Coyote will try and match it to its pre-defined list (Figure 3).

```
cidadetran.bradesco                                    {
                                                           Malware.bank_type = 2;
                                                           flag = true;
                                                       }
                                                       else if (text3 ==
ne12.bradesconetempresa.b.br
                                                           Malware.bank_type = 2;
                                                           flag = true;
                                                       }
                                                       else if (text3 ==
binance.com                                                {
                                                           Malware.bank_type = 3;
                                                           flag = true;
                                                       }
                                                       else if (text3 ==
mercadobitcoin.com.br                                                  {
                                                           Malware.bank_type = 3;
                                                           flag = true;
                                                       }
                                                       else if (text3 ==
bitcointrade.com.br                                                    {
                                                           Malware.bank_type = 3;
                                                           flag = true;
                                                       }
                                                       else if (text3 ==
electrum                                                   {
```

Fig. 3: Coyote attempting to match bank names to its list

The table shows how Coyote classifies the banks and crypto exchanges using their name or web address. In each class, Coyote searches for a number of different addresses; according to our research, there are 75 different addresses.

Name	Type
Banco do Brasil	0
CaixaBank	1
Banco Bradesco	2
Cryptocurrency (Binance, Electrum, bitcoin, Foxbit, and others)	3
Santander	4
Router-app	5
Original bank	6
Sicredi	7
Banco do Nordeste	8
Expanse apps	9

Banks and corresponding number type for Coyote's attempts at matching

Without UIA, parsing the sub-elements of another application is a nontrivial task. To be able to effectively read the contents of sub-elements within another application, a developer would need to have a very good understanding of how the specific target application is structured.

Coyote can perform checks, regardless of whether the malware is online or operating in an offline mode. This increases the chances of successfully identifying a victim's bank or crypto exchange and stealing their credentials.

UIA provides several things for an attacker, including a simple solution for malware developers to parse sub-elements of another application.

Additional UIA tactics, techniques, and procedures

This is the first instance we have seen of a malware using UIA, which indicates how fast malware developers are adopting new techniques into their creations. Note: This is just one example of the potential malicious use of UIA.

Figure 4 shows how UIA can be used not only to identify critical UI components, but also to extract sensitive data from them.

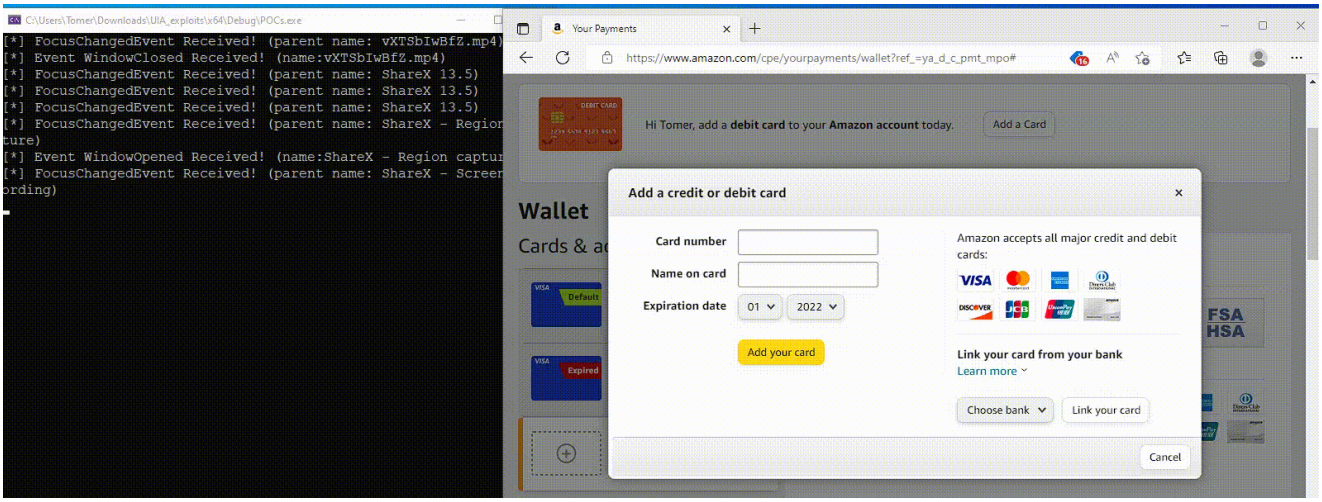


Fig. 4: PoC of UIA abuse to extract sensitive information

In Figure 5, we demonstrate how attackers can manipulate UI components to carry out stealthy social engineering attacks. The attacker alters the browser's address bar and simulates a click, seamlessly redirecting the victim to a malicious server — all with minimal visual indication.

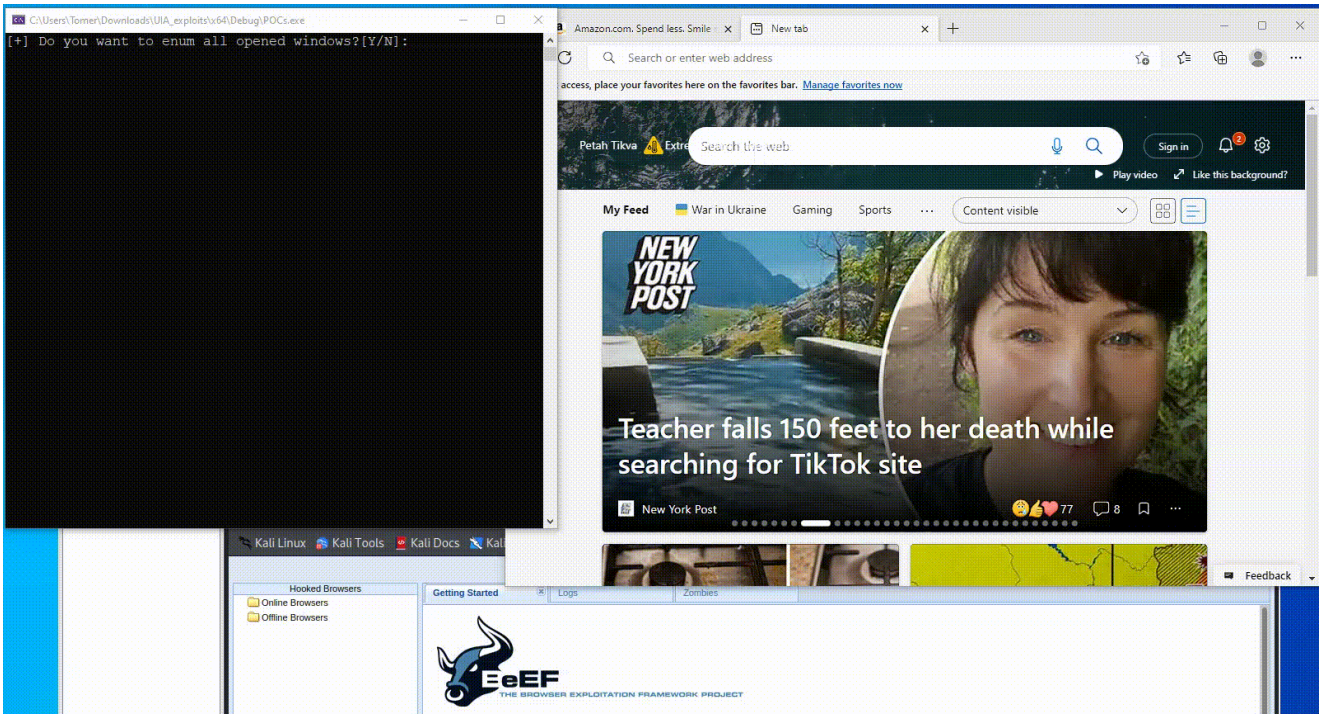


Fig. 5: PoC of abusing UIA for social engineering

For the full details, check out our original UIA blog post.

Detect anomalous UIA use

As for detection of UIA itself, administrators can monitor the use of the *UIAutomationCore.dll*. If it is loaded to a previously unknown process, it should raise legitimate cause for concern.

Similarly, network administrators can monitor the named pipes that are opened on an endpoint by UIA as another indicator of its use. Figure 6 and Figure 7 are osqueries that can be used to detect such activity.

```
  SELECT DISTINCT pid, name, proc.path FROM process_memory_map AS pmm JOIN processes
AS proc USING(pid) WHERE pmm.path LIKE '%uiautomationcore.dll'
```

Fig. 6: Processes that load UIAutomationCore.dll

```
  WITH uia_pipes AS (SELECT name AS pipe_name, SUBSTR(name, 10, INSTR(SUBSTR(name,
10), '_')-1) AS pid FROM pipes WHERE name LIKE 'UIA_PIPE_%' ) SELECT DISTINCT pid,
name AS process_name, path, pipe_name FROM uia_pipes JOIN processes USING(pid)
```

Fig. 7: Processes that opened the UIA named pipe

Akamai Hunt, Akamai's managed threat hunting service, offers its customers protection in the form of a large set of anomaly detection techniques that constantly monitor the environment in an attempt to detect malicious activity. Akamai Hunt customers were scanned to identify anomalous UIA use, and were alerted to any suspicious activity.

Conclusion

Malware is constantly evolving — and in this ongoing game of cat and mouse (or, in this case, coyote and squirrel), it's crucial for both defenders and attackers to stay ahead of the curve by tracking new and emerging threats.

Although UIA may seem like a harmless tool, as we highlighted in our previous blog post, abusing its capabilities can lead to serious damage for organizations. By exposing Coyote's tactics, we hope defenders will be better equipped with multiple ways to detect and respond to this threat.

We believe UIA represents a viable and dangerous attack vector that warrants serious attention — and one we're likely to see increased abuse of in the future.

Special thanks to @johnk3r for initially bringing this malware to our attention.

Read more research

Written by

Tomer Peled

July 22, 2025

Written by

Tomer Peled

Tomer Peled is a Security Researcher at Akamai. In his daily job, he conducts research ranging from vulnerability research to OS internals. In his free time, he likes to cook, do Krav Maga, and game on his PC.