White Paper

McAfee®

Ten Days of Rain

Expert analysis of distributed denial-of-service attacks targeting South Korea

Table of Contents

Executive Summary

On March 4, 2011 McAfee began detecting distributed denial-of-service (DDoS) activity against targets in South Korea. The attacks were sourced from a botnet architecture leveraging compromised hosts in South Korea. The DDoS attacks were targeting South Korean government websites as well as the network of U.S. Forces Korea (USFK). In addition to the DDoS attacks, which were successful in negatively impacting the availability of multiple Korean targets, McAfee also analyzed the malware responsible for initially turning the attacking hosts into bots, and thus placing them under the command and control of the botnet operator. While the attack itself seems fairly generic at first glance, there are several things that make this particular combination of targets, malware, and botnet activity different from many we've analyzed, warranting our investigation.

The DDoS attacks had clearly defined targets and a finite window of operation preconfigured as 10 days. Once that time expired, the bot on the compromised hosts would halt DDoS activity and render the host inoperable, thus requiring a full rebuild of operating systems, applications, and user data. While highly destructive code like this was common with early malware, it has long since given way to bots that allow for long-term command and control. Cybercriminals realized that compromised computers under their full control are much more valuable to them for sending spam, proliferating malware, and for harvesting valuable data from the compromised device. While there is some temporary satisfaction from an act of vandalism that renders the machine inoperable, this outcome has given way to financial motivations. The bots in these attacks, however, were configured to only perform DDoS attacks instead of having multiple capabilities and allowing for a wider range of nefarious uses.

There was a high degree of cryptographic diversity: many disparate algorithms were utilized with the goal of slowing analysis and ultimately increasing time to mitigation. In addition to the bots themselves, a multitier botnet architecture, optimized to mitigate takedowns, was employed to ensure operational resiliency. In short, several steps were taken to ensure that the mission was executed without interruption, within the predefined attack window—and following, ensuring that all vehicles of attack would be destroyed, thus limiting forensic analysis.

Working very closely with customers and partners in the public and private sector that have assets in South Korea, we were able to detect, reverse engineer, and mitigate the attacks. Leveraging various McAfee solutions, outlined in Appendix A, both the malware and DDoS attacks could be addressed. We also tracked the botnet controllers, outlined in Appendix B, and released a free tool to McAfee and non-McAfee customers to remove the malware.

Beyond the threat mitigation, the questions of how, who, and why still remain. The remainder of this paper will explore the technical details behind the attacks along with perspectives regarding possible actors and motivations.

High-Level Attack Overview

Like most distributed attacks, these attacks start with compromised hosts. The initial compromises could have occurred through any number of traditional vehicles designed to get malicious code on a host. One of the likely sources was a South Korean file-sharing site. Once the host has been infected with the bot, the host could be controlled by a remote command and control, or C&C, server. We discovered several C&C servers on hosts that were probably previously compromised targets themselves. The C&C servers were distributed across multiple geographies. The list of first-tier C&C servers we observed can be found in Appendix B and a diagram of the geographic distribution is illustrated in Figure 1.

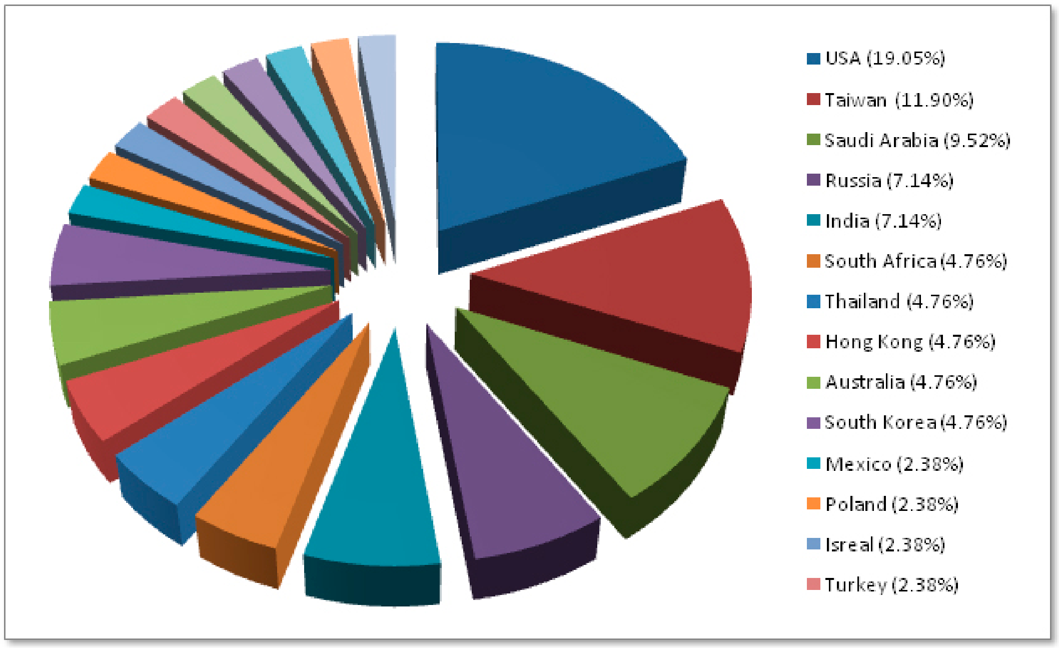


Figure 1. Geographical distribution of C&C servers.

The C&C architecture was multitier, allowing for greater scalability and takedown resiliency. Having a multitier architecture is computer science 101; if you want to make something scalable, you build a hierarchy. In design, this is quite similar to the way DNS servers operate—by having various name server tiers. At the top, you have the root servers that tell you where to locate the servers responsible for top-level domains such as .COM, .NET, .GOV, .KR and so on. For the bots, an infected host communicated with one or many first-tier C&C servers. If one C&C server got removed, perhaps as part of takedown activity initiated by various government, telecommunication, academic, or large enterprises, the bot could simply communicate with another first-tier C&C server. Additionally, the first-tier servers were essentially only a redirector to the second-tier C&C servers. These second-tier servers were ultimately in control, and because of the first-tier servers, the second-tier servers were safer from detection and takedowns. In addition to having a second tier, this model can and may have, in this instance, been expanded to a third-tier architecture providing even greater takedown resiliency.

The bots took advantage of multiple encryption ciphers in an attempt to make static analysis more difficult and force the reverse engineering process to rely on dynamic analysis. The cryptographic algorithms we identified included: RC4, AES, RSA, and MD5 hashing. Various encryption algorithms were used throughout multiple areas within each bot.

With an army of bots now controlled by the various C&C servers, the DDoS attacks could commence. The targets of the attack are listed in Appendix C. There are a significant number targets related to the South Korean military. Some additional targets included USFK (U.S. Forces Korea), and there were some non-military South Korean targets. The DDoS attacks utilized ICMP and UDP messages and HTTP requests as their attack vector. The bots in this case were only configured to perform DDoS. Further limiting their capabilities was a predetermined start and stop time for the attack that would last only 10 days. While the botnet configuration was set to 10 days, if desired, a configuration update could have been sent to extend the operational window.

At the conclusion of the "Ten days of Rain" DDoS attacks, the bots were configured to self-destruct. This was accomplished by deleting and overwriting key files and finally damaging the master boot record, or MBR. This is known as an MBR Flash, and it essentially makes the host unusable to the legitimate owner and the C&C servers.

Detailed Attack Overview

Analysis

Unlike many other botnets, the malware installed as these C&C clients lacked command interpreter functionality. This results in very limited flexibility in how the bots are used. As illustrated in Figure 2, it was a simple downloader Trojan that would download encrypted file packages from its C&C servers, decrypt them based on a method denoted in the package, and potentially launch them if they contained executable code. The following analysis is based on the payloads that we observed during the attacks

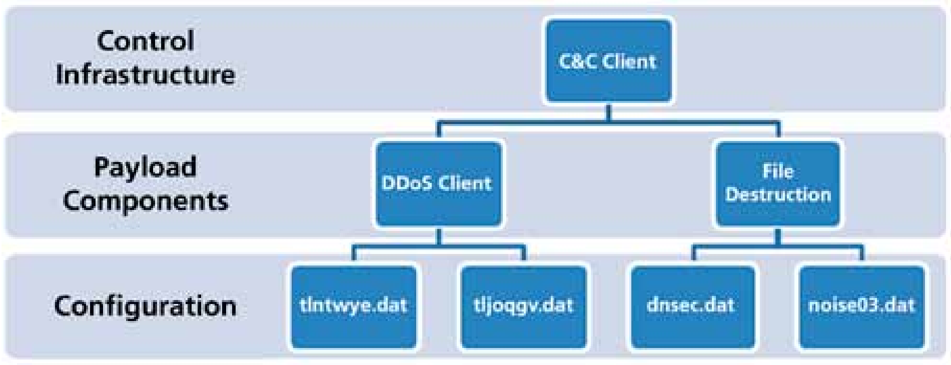


Figure 2. Trojan architecture.

Payload binaries

These payload binaries were downloaded during the course of the attacks and did not contain hardcoded configurations, but rather, they were dependent on configuration files. Instead of simply downloading the configuration files, the attackers chose to distribute more dropper binaries that would place the configuration files in the system. This made it impossible to facilitate automated botnet tracking once the file format was identified, as any change in configuration involved the execution of a new malware binary that writes the new configuration file. By analyzing the different types of malware binaries observed during the course of the attacks, we were able to glean detailed information about the various attack components and how they operated including the:

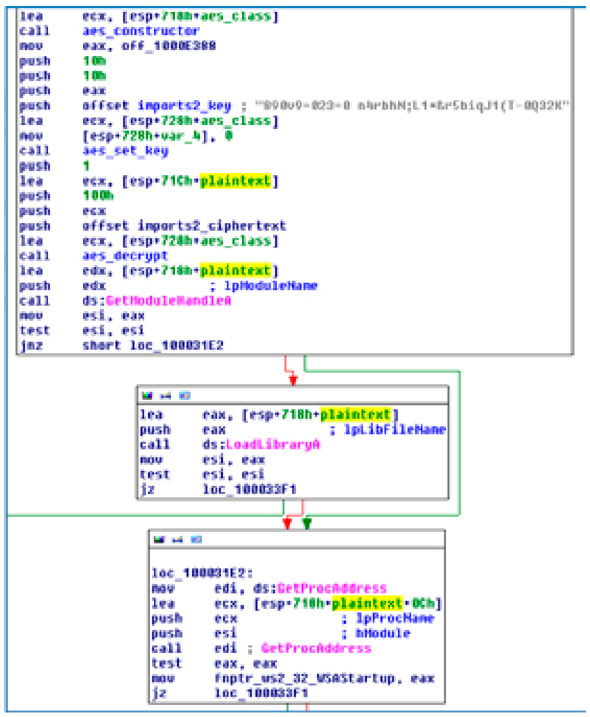
- Command and control client
- DDoS component
- Date monitor with self destruct code

Let’s examine each in detail.

Command and control client

The most critical binary for the attackers was the command and control client service. This service was required to maintain control of the infected computers. It was implemented as a service DLL that was loaded into svchost.exe. The observed binary had the MD5 a63f4c213e2ae4d6caa85382b65182c8.

As shown in Figure 3, the malware decrypts its imports with a static 128bit AES key. This obfuscation technique hardly thwarts human analysis. The decryption can be executed, and the imported function pointers can be annotated manually afterwards. Thus, it is likely effective against anti-virus emulators because of the amount of virtual CPU cycles spent on decryption. As such, proactive detection that is based purely on emulation was likely circumvented.



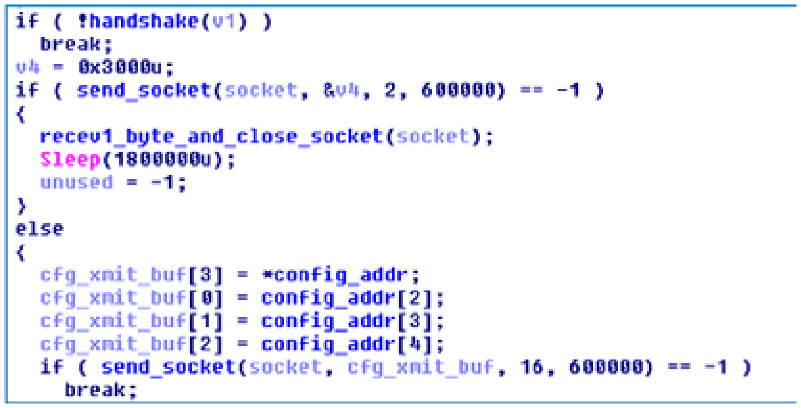
```
lea     ecx, [esp+718h+aes_class]
call    aes_constructor
mov     eax, off_1000E388
push    10h
push    10h
push    eax
push    offset imports2_key ; "890v9-823-0 n4rbhN;L1+&r5biqJ1(T-8Q32K"
lea     ecx, [esp+728h+aes_class]
mov     [esp+728h+var_4], 0
call    aes_set_key
push    1
lea     ecx, [esp+71Ch+plaintext]
push    100h
push    ecx
push    offset imports2_ciphertext
lea     ecx, [esp+728h+aes_class]
call    aes_decrypt
lea     edx, [esp+718h+plaintext]
push    edx             ; lpModuleName
call    ds:GetModuleHandleA
mov     esi, eax
test    esi, esi
jnz     short loc_100031E2

lea     eax, [esp+718h+plaintext]
push    eax             ; lpLibFileName
call    ds:LoadLibraryA
mov     esi, eax
test    esi, esi
jz      loc_100033F1

loc_100031E2:
mov     edi, ds:GetProcAddress
lea     ecx, [esp+718h+plaintext+0Ch]
push    ecx             ; lpProcName
push    esi             ; hModule
call    edi ; GetProcAddress
test    eax, eax
mov     fnptr_ws2_32_WSAStartup, eax
jz      loc_100033F1
```

Figure 3. Decryption process.

The code then loads the C&C configuration file faultrep.dat from the system32 folder. While the C&C application also decrypts the configuration's filename with 128-bit AES, the initial dropper contains this filename in plain text. This design hints at multiple authors that were not all aware of this filename being encrypted in other parts of this attack. This configuration file contains a list of C&C server IPs and ports the bot uses for making network connections. While the bot supports arbitrary ports for the C&C servers, the observed configuration files only contained C&C servers listening in on port 443. The C&C client then picks one of these servers randomly until it can successfully connect.

The application layer handshake as shown in Figure 4 works as follows:

- Sending the magic 45196327h in host byte order
- Receiving four bytes and checking if it is the same magic
- Sending another constant value of 3000h in host byte order
- Sending the first four DWORDs of the configuration file in shuffled order



```
if ( !handshake(v1) )
  break;
v4 = 0x3000u;
if ( send_socket(socket, &v4, 2, 600000) == -1 )
{
  recev1_byte_and_close_socket(socket);
  Sleep(1800000u);
  unused = -1;
}
else
{
  cfg_xmit_buf[3] = *config_addr;
  cfg_xmit_buf[0] = config_addr[2];
  cfg_xmit_buf[1] = config_addr[3];
  cfg_xmit_buf[2] = config_addr[4];
  if ( send_socket(socket, cfg_xmit_buf, 16, 600000) == -1 )
    break;
```

Figure 4. Application layer handshake.

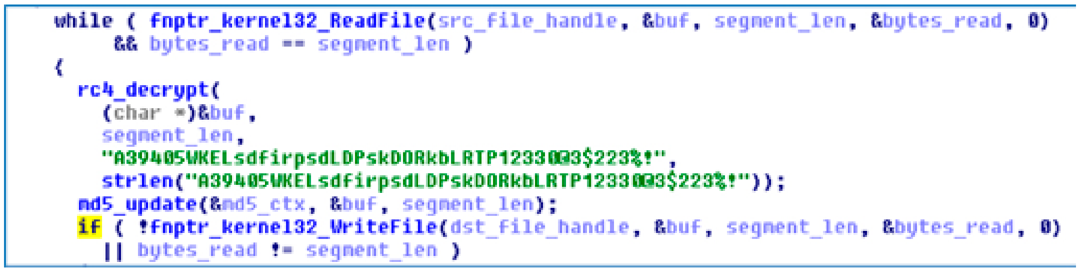
Following the handshake, the client will download an arbitrary number of files from the host. Notably, the C&C server provides a filename. The client opens the file even if the file is already present in the system. If the file is already present, it will send the current file size to the C&C and append any data to the existing file. The whole communication in this stage is encrypted with a simple XOR key of 5e9adaf35a6b284b. The downloaded files are all assigned the same, randomly generated timestamp, apparently in an effort to further thwart file system forensics.

The next step is for the downloaded package files to be parsed. The headers of these files are encrypted with RSA; the 128-bytes long private key resides within the binary. The downloaded files support three different actions to be executed on the client:

1. Extract the executable and launch it using CreateProcess.
2. Extract the executable and launch it using CreateProcess without further actions.
3. Extract the executable and launch it using CreateProcess while invoking a specific application programming interface (API) from an unknown DLL. The name of the DLL and the API are AES encoded with both key and ciphertext residing statically in the binary. The same key used for decrypting the other imports is being used. However, decrypting the given ciphertext yields garbage data. We did not observe this functionality being used in the configurations we have observed.
4. Extract a command line for an existing binary, and launch it using CreateProcess.

The payloads are decrypted with RC4 using a static key in the binary and the content is integrity checked with MD5 hash. This is shown in Figure 5. There is no signature scheme being used, and one downloaded package file can contain multiple such payloads. We implemented a real-time monitoring tool for this C&C mechanism that monitored all updates being distributed by the C&C servers. This allowed us to observe payloads and gather additional information about the attack internals.

In addition to the command and control client, we investigated the components responsible for the DDoS.



```
while ( fnptr_kernel32_ReadFile(src_file_handle, &buf, segment_len, &bytes_read, 0)
     && bytes_read == segment_len )
{
  rc4_decrypt(
    (char *)&buf,
    segment_len,
    "A39405WKELsdfirpsdLDPskDORkbLRTP12330@3$223%!",
    strlen("A39405WKELsdfirpsdLDPskDORkbLRTP12330@3$223%!"));
  md5_update(&md5_ctx, &buf, segment_len);
  if ( !fnptr_kernel32_WriteFile(dst_file_handle, &buf, segment_len, &bytes_read, 0)
    || bytes_read != segment_len )
```

Figure 5. Decrypted payload.

Distributed denial-of-service component

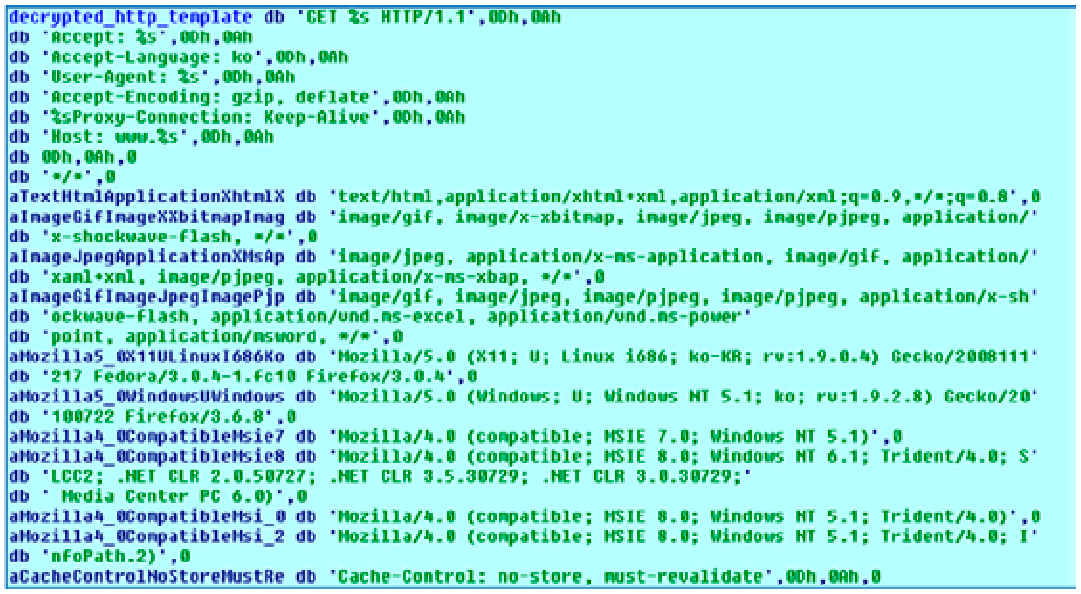
The downloaded DDoS component is also implemented as service DLL; the specific MD5 we observed was 0a21b996e1f875d740034d250b878884. Instead of receiving commands directly from the C&C server, it parses the two local configuration files tljoqgv.dat and tlntwye.dat periodically. The names of these files are obfuscated; they are stored with each character value before the file extension being bytewise decremented.

- The first configuration file, tlntwye.dat, is an eight-byte configuration file that contains the timestamp of the next DDoS start encoded as a floating-point variant time. If this file is not present, the DDoS will start right away. Otherwise, the DDoS component will check every 10 minutes to see if the attack can start now.

- The second configuration file, tljoqgv.dat, is an encrypted list of DDoS targets. These targets can be resolved either by IP address directly or by DNS. The configuration files we observed consisted only of DNS name targets. A separate thread is continuously resolving these targets to IP address during the whole attack run.

The DDoS client supports four different methods of attack:

1. Method one is the repeated issuing of DNS queries for the given DNS name list. This utilizes invocation of the Microsoft Windows DnsQuery API from dnsapi.dll. The sub-function that implements the DNS-based DDoS contains a fallback code path taken if the host name list is empty; this code path is using a static DNS name target list from the decrypted resources that consists of google.com, yahoo.com, naver.com, daum.net, and microsoft.com.
2. Method two, as shown in Figure 6, is an HTTP request flooding DDoS that assembles random requests based on an encrypted template within the resource section. The "User-Agent" and "Accept" headers of the request are chosen by picking a random entry from a list of possible selections within the encrypted resources. It should be noted that the "Accept-Language" is hardcoded to Korean:
3. Method three resembles a classic UDP flood by sending packets with random content.
4. Method four resembles a classic PING flood by sending ICMP packets with random content.



```
decrypted_http_template db 'GET %s HTTP/1.1',0Dh,0Ah
db 'Accept: %s',0Dh,0Ah
db 'Accept-Language: ko',0Dh,0Ah
db 'User-Agent: %s',0Dh,0Ah
db 'Accept-Encoding: gzip, deflate',0Dh,0Ah
db '%sProxy-Connection: Keep-Alive',0Dh,0Ah
db 'Host: www.%s',0Dh,0Ah
db 0Dh,0Ah,0
db '*/*',0
aTextHtmlApplicationXhtmlX db 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',0
aImageGifImageXXbitmapImag db 'image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/'
db 'x-shockwave-flash, */*',0
aImageJpegApplicationXMsAp db 'image/jpeg, application/x-ms-application, image/gif, application/'
db 'xaml+xml, image/pjpeg, application/x-ms-xbap, */*',0
aImageGifImageJpegImagePjp db 'image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/x-sh'
db 'ockwave-flash, application/vnd.ms-excel, application/vnd.ms-power'
db 'point, application/msword, */*',0
aMozilla5_0X11ULinuxI686Ko db 'Mozilla/5.0 (X11; U; Linux i686; ko-KR; rv:1.9.0.4) Gecko/2008111'
db '217 Fedora/3.0.4-1.fc10 Firefox/3.0.4',0
aMozilla5_0WindowsUWindows db 'Mozilla/5.0 (Windows; U; Windows NT 5.1; ko; rv:1.9.2.8) Gecko/20'
db '100722 Firefox/3.6.8',0
aMozilla4_0CompatibleMsie7 db 'Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)',0
aMozilla4_0CompatibleMsie8 db 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4.0; S'
db 'LCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729;'
db ' Media Center PC 6.0)',0
aMozilla4_0CompatibleMsi_0 db 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)',0
aMozilla4_0CompatibleMsi_2 db 'Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; I'
db 'nfoPath.2)',0
aCacheControlNoStoreMustRe db 'Cache-Control: no-store, must-revalidate',0Dh,0Ah,0
```

Figure 6. HTTP request.

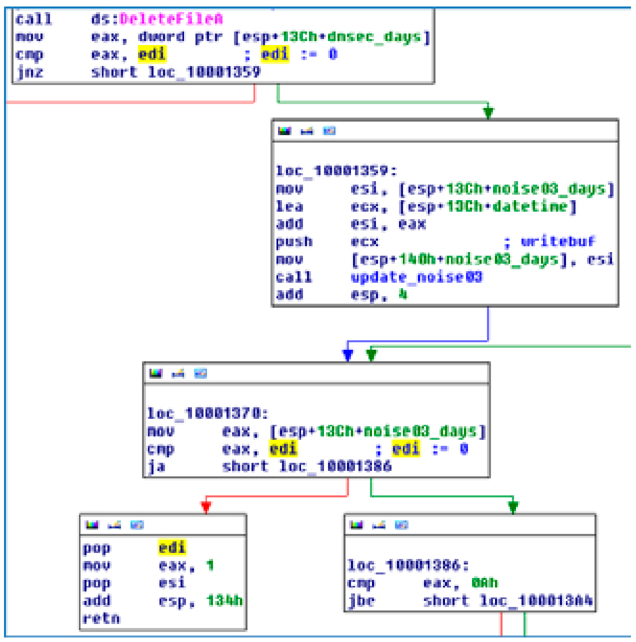
After performing one DDoS run, the component's code loops and begins to check for a timestamp configuration file again. Beyond the DDoS component itself are the specific components responsible for cryptographic ciphers.

While the resources in the DDoS component are encrypted with a simple eight-byte XOR key, the configuration is encrypted with a considerably more elaborate algorithm. While this encryption does not pose a significant hurdle for an analyst with possession of this binary because it uses a static key within the binary and all decrypted configuration information can be dumped and decrypted from memory, it does hinder automation of attack target tracking.

Date monitor and self-destruct payload

Another service DLL with MD5 c963b7ad7c7aefbe6d2ac14bed316cb8 continuously monitors the current date of the local clock on the compromised host and initiates a self-destruct function upon certain conditions. This payload uses two configuration files, namely noise03.dat and dnsec.dat.

- The noise03.dat file is initially created by the first dropper and contains the install timestamp as floating-point variant. This initial timestamp is never changed by any other components. It additionally contains an integer number of days to live, as offset from the installation timestamp. The dropper initializes this to zero.
- The dnsec.dat file can be created by the C&C client or any downloaded component and will be read by the monitoring component. Illustrated in Figure 7, it contains the number of days to add to the existing time to live as a four-byte integer, similar to an "insert coins to continue playing" mechanism. However, if the amount of days is set to zero, the days to live offset in noise03.dat is reset to zero as well.



```
call    ds:DeleteFileA
mov     eax, dword ptr [esp+13Ch+dnsec_days]
cmp     eax, edi            ; edi := 0
jnz     short loc_10001359

loc_10001359:
mov     esi, [esp+13Ch+noise03_days]
lea     ecx, [esp+13Ch+datetime]
add     esi, eax
push    ecx                 ; writebuf
mov     [esp+140h+noise03_days], esi
call    update_noise03
add     esp, 4

loc_10001370:
mov     eax, [esp+13Ch+noise03_days]
cmp     eax, edi            ; edi := 0
ja      short loc_10001386

pop     edi
mov     eax, 1
pop     esi
add     esp, 134h
retn

loc_10001386:
cmp     eax, 0Ah
jbe     short loc_100013A4
```

Figure 7. Monitoring component.

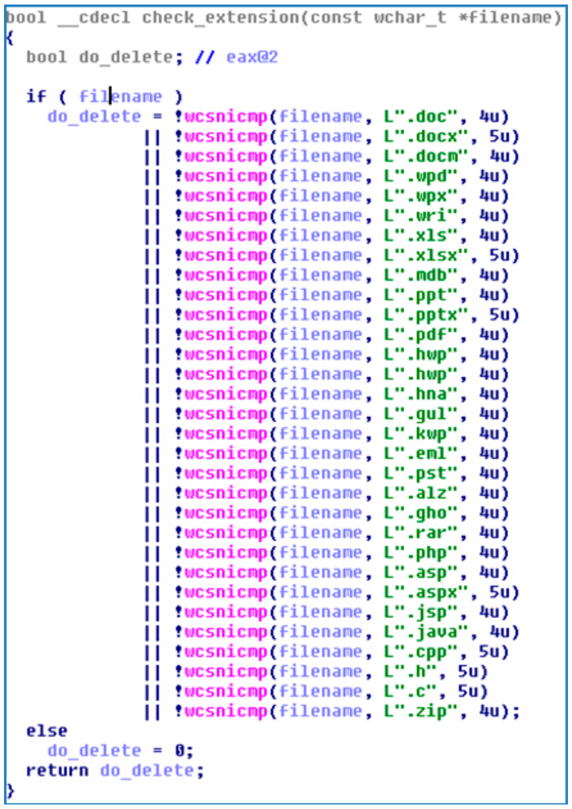
This component also periodically checks the current system time. It then starts to destroy the infected computer if one of the following three conditions is met:

1. The current system time has been modified to be before the initial infection timestamp as recorded within noise03.dat. However, it is still possible to modify the system clock backwards if you take care not to do it before the recorded infection timestamp.
2. The current system time is after the infection timestamp plus the amount of days given to live by any dnsec.dat additions.
3. The time to live timestamp has been reset to zero by a zero dnsec.dat file.

Note that Figure 7 shows the total amount of overall days to live is hardcoded to be 10 days in total; therefore, the system will be destroyed after at most 10 days after initial infection. This is how we derived the operation timeline to have a maximum of 10 days.

The destruction payload then recursively enumerates all files and folders, illustrated in Figure 8, on all fixed drives and tries to find specific file extensions. The code to check file extensions suffers from some mistakes due to copy and paste; for example, not only .java but .javanything files will be deleted. If the file has the desired file extension, it will be first overwritten with zeroes. Interestingly, the code then utilizes a huge C++ CAB file implementation to create a new CAB file per overwritten file and adds the already zeroed-out file to the CAB. This is another indicator of multiple engineers working on this codebase without everyone understanding the entirety of the code. The code then deletes the original overwritten file.

A parallel thread overwrites the start of every physical drive in the computer with zeroes. This effectively renders the computer unusable, as the master boot record (MBR) will be overwritten, and the computer will be unable to boot. Hence, the combination of these two destruction techniques causes maximum havoc on the compromised host and its users. This is the only component we analyzed that didn't employ cryptographic ciphers or import obfuscation.



```
bool __cdecl check_extension(const wchar_t *filename)
{
  bool do_delete; // eax@2

  if ( filename )
    do_delete = !wcsnicmp(filename, L".doc", 4u)
             || !wcsnicmp(filename, L".docx", 5u)
             || !wcsnicmp(filename, L".docm", 4u)
             || !wcsnicmp(filename, L".wpd", 4u)
             || !wcsnicmp(filename, L".wpx", 4u)
             || !wcsnicmp(filename, L".wri", 4u)
             || !wcsnicmp(filename, L".xls", 4u)
             || !wcsnicmp(filename, L".xlsx", 5u)
             || !wcsnicmp(filename, L".mdb", 4u)
             || !wcsnicmp(filename, L".ppt", 4u)
             || !wcsnicmp(filename, L".pptx", 5u)
             || !wcsnicmp(filename, L".pdf", 4u)
             || !wcsnicmp(filename, L".hwp", 4u)
             || !wcsnicmp(filename, L".hwp", 4u)
             || !wcsnicmp(filename, L".hna", 4u)
             || !wcsnicmp(filename, L".gul", 4u)
             || !wcsnicmp(filename, L".kwp", 4u)
             || !wcsnicmp(filename, L".eml", 4u)
             || !wcsnicmp(filename, L".pst", 4u)
             || !wcsnicmp(filename, L".alz", 4u)
             || !wcsnicmp(filename, L".gho", 4u)
             || !wcsnicmp(filename, L".rar", 4u)
             || !wcsnicmp(filename, L".php", 4u)
             || !wcsnicmp(filename, L".asp", 4u)
             || !wcsnicmp(filename, L".aspx", 5u)
             || !wcsnicmp(filename, L".jsp", 4u)
             || !wcsnicmp(filename, L".java", 4u)
             || !wcsnicmp(filename, L".cpp", 5u)
             || !wcsnicmp(filename, L".h", 5u)
             || !wcsnicmp(filename, L".c", 5u)
             || !wcsnicmp(filename, L".zip", 4u);
  else
    do_delete = 0;
  return do_delete;
}
```

Figure 8. Destruction payload.

While conducting our analysis, we generated solutions to mitigate the threat while forming theories about the attackers and motives.

Remediation

McAfee worked very closely with various public and private sector customers and partners in South Korea. Following our analysis we released a stinger—which is freely available to McAfee and non-McAfee customers alike, and updated with purpose-built malware removal code to address these attacks. We also tracked the botnet sources within the McAfee Global Threat Intelligence™ cloud, updated our anti-virus .DATs, and other solutions, and made these available to our customers. These solutions were successful in removing the malware, preventing the malware from destroying files, and preventing the flashing of the MBR.

Perspectives

Comparison with July 4, 2009 DDoS attacks

These attacks very closely resemble DDoS attacks that began on July 4, 2009, Independence Day in the United States. The 2009 attacks used a botnet of more than 150,000 machines predominately concentrated in South Korea. The botnet launched a DDoS attack against almost four dozen targets in South Korea and the United States. For a complete list of the 2009 targets, see Appendix D, and note that 14 of the targets were the same across the 2009 and 2011 attacks.

On July 10, 2009, the botnet updated itself with destructive components that proceeded to delete user data files such as those with the following extensions: .xml, .xls, .ppt, .doc, .pdf, .c, .cpp, among others. The botnet also compressed the files prior to deletion with a gzip algorithm and encrypted them with a password. This capability was also present in the 2011 code but not activated.

Other details about the 2009 attack include:

- It overwrote the first 512 bytes of each attached storage device with a string beginning with "Memory of Independence Day" wiping out the MBRs and volume boot records on infected botnet machines
- On July 8, the botnet began sending spam messages with the subject "Memory Of ..." from address "Independence" and the word "last" in the body. It used a Korean character set and had a memory.rar attachment file, which was empty.
- The 2011 attacks took place exactly 20 months to the day after the 2009 attacks

Based on the similarities between the 2011 and 2009 attacks, we believe that there is strong, although circumstantial, evidence to conclude that both attacks had originated from the same adversary. While there are many similarities, there are also differences.

The sophistication of the 2011 attack increased with the introduction of multiple encryption algorithms and more advanced C&C capabilities. Also, during the 2011 attacks, the United States was only moderately impacted by the DDoS. Only two U.S. sites were targeted, including usfk.mil and kunsan.af.mil. The majority of the 2011 attacks were focused on South Korea. Associated with the 2009 attacks, Figure 9 illustrates the global distribution of the botnets by IP address geolocation. Clearly, the majority of bots were located in South Korea, with some others in the United States and Western Europe. Figure 10 simply shows that the most of the DDoS traffic volume was coming out of South Korea.

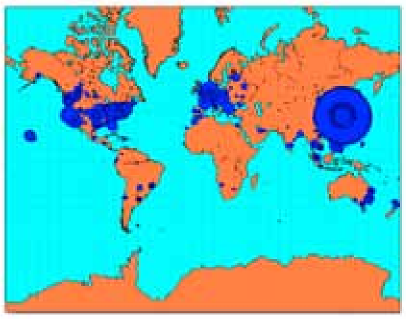


Figure 9. 2009 geolocation of bots by IP address.

Figure 10. 2009 DDoS traffic volume.

Perspective summary

The level of technical sophistication behind Ten Days of Rain, being used for the relatively simplistic act of a DDoS attack, doesn't track. Why was so much cryptographic work utilized? Why was a multitier architecture, designed to be so resilient to takedowns, used if the operational life of the bots was only 10 days? Why not keep control of the compromised hosts; why not utilize those systems for future tasks instead of self-destructing?

This wasn't a surgical strike; it was more like a sledgehammer, as most DDoS attacks are. As such, it was noisy, making it easier to detect than a stealthy attack that might be used to steal sensitive data. Knowing this, the attackers relied on the encryption to buy them more time against reverse engineering until the DDoS attack window expired. But what was their motivation? A number of theories can be entertained to address this question, and while a definitive answer isn't always available, based on our technical analysis and investigation, we feel that the following scenario captures the likely actors and motives behind these attacks.

This attack was engineered by multiple individuals with varying insight into the overall architecture of the code. This may have been a test of South Korea's preparedness to mitigate cyberattacks, possibly by North Korea or their sympathizers. While the code and botnet architecture were advanced, the attack itself was very limited and may have been utilized to test and observe how quickly the attack would be discovered, reverse engineered, and mitigated. Armed with this knowledge, the aggressor could launch cyberattacks, possibly in conjunction with kinetic attacks, with a greater understanding of South Korea's incident response capabilities. As such, the attackers could better understand their own requirements for a successful campaign.

Conclusion

DDoS, malware-leveraging encryption, and multitier botnet architectures are not new. Nor are attacks against South Korea that suspiciously align with North Korea's agenda. However, the combination of technical sophistication juxtaposed with relatively limited execution and myopic outcome is analogous to bringing a Lamborghini to a go-cart race. As such, the motivations appear to outweigh the attack, making this truly seem like an exercise to test and observe response capabilities.

Credits and Acknowledgements

As with most initiatives at McAfee, this was a team effort bringing together engineers from McAfee Labs with other departments at McAfee, our partners, and our customers. We would like to give a special thanks to the US-CERT, Department of Defense analysts, and AhnLabs as well as our own—Georg Wicherski, Dmitri Alperovitch, Brian Contos—and countless others for their tireless effort, support, and fighting the good fight every day.

Appendix A: McAfee Solutions

Here are just a few of the McAfee solutions that could be leveraged to help mitigate various aspects of this attack.

Protection from malware

- McAfee Application Control
- McAfee Configuration Control
- McAfee VirusScan® Enterprise
- McAfee Vulnerability Manager and Policy Auditor
- McAfee Web Gateway
- McAfee Network Threat Response
- McAfee Risk Advisor
- McAfee Global Threat Intelligence™

Protection from the DDoS attacks

- McAfee Network Security Platform
- McAfee Enterprise Firewall
- McAfee Global Threat Intelligence

Appendix B: Observed First-Tier C&C Servers

Host	Country
173-11-235-222-houston.txt.hfc.comcastbusiness.net (173.11.235.222)	United States
208.36.53.174.ptr.us.xo.net (208.36.53.174)	United States
dial-242.r1.scccrk-gwy.infoave.net (206.74.76.243)	United States
32.106.118.196	United States
static-ip-208.71.147.242.airlogic.net (208.71.147.242)	United States
63.163.221.71	United States
static-ip-208.71.147.242.airlogic.net (208.71.147.242)	United States
63.163.221.71	United States
119-15-208-97.veetime.com (119.15.208.97)	Taiwan
59-125-224-43.hinet-ip.hinet.net (59.125.224.43)	Taiwan
59-120-179-11.hinet-ip.hinet.net (59.120.179.11)	Taiwan
59-125-224-43.hinet-ip.hinet.net (59.125.224.43)	Taiwan
59-120-179-11.hinet-ip.hinet.net (59.120.179.11)	Taiwan
212.62.100.211	Saudi Arabia
shabnet5-42.shabakah.net (212.102.5.42)	Saudi Arabia
212.62.100.211	Saudi Arabia
shabnet5-42.shabakah.net (212.102.5.42)	Saudi Arabia
h86-62-115-242.ln.rinet.ru (86.62.115.242)	Russia
212.58.215.77	Russia
host-88-215-130-6.stv.ru (88.215.130.6)	Russia
abts-north-static-085.36.176.122.airtelbroadband.in (122.176.36.85)	India
203.196.252.244	India
203.196.252.244	India
dsl-241-141-76.telkomadsl.co.za (41.241.141.76)	South Africa
113-53-236-67.totisp.net (113.53.236.67)	Thailand

203186126225.static.ctinets.com (203.186.126.225)	Hong Kong
61-91-86-34.static.asianet.co.th (61.91.86.34)	Thailand
210.0.204.175	Hong Kong
gateway2.liveip.co.za (196.23.164.39)	South Africa
growli.lnk.telstra.net (120.151.118.10)	Australia
147.175.129.216	Slovakia
growli.lnk.telstra.net (120.151.118.10)	Australia
147.175.129.216	Slovakia
ip-201-168-56-139.marcatel.net.mx (201.168.56.139)	Mexico
bilon057.miks.uj.edu.pl (149.156.160.60)	Poland
194.90.168.146	Israel
dsl95.9-28681.static.ttnet.net.tr (95.9.112.9)	Turkey
91.74.106.98	United Arab Emirates
uu212-190-216-147.unknown.uunet.be (212.190.216.147)	Belgium
p1001-ipbf2503sapodori.hokkaido.ocn.ne.jp (114.167.76.1)	Japan
cable-static-44-108.intergga.ch (157.161.44.108)	Switzerland
host153-234-static.22-87-b.business.telecomitalia.it (87.22.234.153)	Italy

Appendix C: Alphabetical List of DDoS Targets

- ahnlab.com
- airforce.mil.kr
- army.mil.kr
- assembly.go.kr
- auction.co.kr
- customs.go.kr
- cwd.go.kr
- daishin.co.kr
- dapa.go.kr
- daum.net
- dcinside.com
- dema.mil.kr
- fsc.go.kr
- gmarket.co.kr
- hanabank.com
- hangame.com
- jcs.mil.kr
- jeilbank.co.kr
- kbstar.com
- kcc.go.kr
- keb.co.kr
- khnp.co.kr
- kisa.or.kr
- kiwoom.com
- korail.com
- korea.go.kr
- kunsan.af.mil
- mnd.mil.kr
- mofat.go.kr
- mopas.go.kr
- naver.com
- navy.mil.kr
- nis.go.kr
- nonghyup.com
- nts.go.kr
- police.go.kr
- shinhan.com
- unikorea.go.kr
- usfk.mil
- wooribank.com

Appendix D: Alphabetical List of DDoS Targets in 2009 Attacks

- banking.nonghyup.com
- blog.naver.com
- ebank.keb.co.kr
- ezbank.shinhan.com
- finance.yahoo.com
- mail.daum.net
- mail.naver.com
- mail.paran.com
- nking.nonghyup.com
- travel.state.gov
- www.ahnlab.com
- www.altools.co.kr
- www.amazon.com
- www.assembly.go.kr
- www.auction.co.kr
- www.chosun.com
- www.defenselink.mil
- www.dhs.gov
- www.dot.gov
- www.egov.go.kr
- www.faa.gov
- www.ftc.gov
- www.hanabank.com
- www.hannara.or.kr
- www.ibk.co.kr
- www.kbstar.com
- www.marketwatch.com
- www.mnd.go.kr
- www.mofat.go.kr
- www.nasdaq.com
- www.ncsc.go.kr
- www.nsa.gov
- www.nyse.com
- www.president.go.kr
- www.site-by-site.com
- www.state.gov
- www.usauctionslive.com
- www.usbank.com
- www.usfk.mil
- www.usps.gov
- www.ustreas.gov
- www.voa.gov
- www.voanews.com
- www.washingtonpost.com
- www.whitehouse.gov
- www.wooribank.com

About McAfee

McAfee, a wholly owned subsidiary of Intel Corporation (NASDAQ:INTC), is the world's largest dedicated security technology company. McAfee delivers proactive and proven solutions and services that help secure systems, networks, and mobile devices around the world, allowing users to safely connect to the Internet, browse, and shop the web more securely. Backed by its unrivaled global threat intelligence, McAfee creates innovative products that empower home users, businesses, the public sector, and service providers by enabling them to prove compliance with regulations, protect data, prevent disruptions, identify vulnerabilities, and continuously monitor and improve their security. McAfee is relentlessly focused on constantly finding new ways to keep our customers safe. http://www.mcafee.com

McAfee
2821 Mission College Boulevard
Santa Clara, CA 95054
888 847 8766
www.mcafee.com