# Revisiting The Bunitu Trojan

blog.malwarebytes.com/threat-analysis/2015/07/revisiting-the-bunitu-trojan/

hasherezade July 13, 2015

This post describes the infection process of the latest version of the **Bunitu Proxy Trojan** as seen delivered by the **Neutrino Exploit Kit** via a malvertising campaign.

We will start from a high-level overview of the issue and used elements. Then, we will dive deeper in the used techniques of hiding and obfuscating the modules.

## What is Bunitu Proxy and why is it dangerous?

As its name suggests, Bunitu Proxy is a Trojan that exposes the infected computer to be used as a proxy for remote clients. It is done in a few steps:

1. Installs itself on the machine
2. Opens ports for the remote connections
3. Registers itself in the remote server (clients database) informing about its address and open ports
4. Accepts connections coming on the exposed ports and bypasses the traffic

It may have various consequences for the infected user. Basically, it uses his/her resources and slows down the network traffic. But it may also frame him/her in some illegal activities carried by the attackers due to the fact that the infected client's IP is the one visible from the

outside.

***Read more: Who's Behind Your Proxy? Uncovering Bunitu's Secrets***

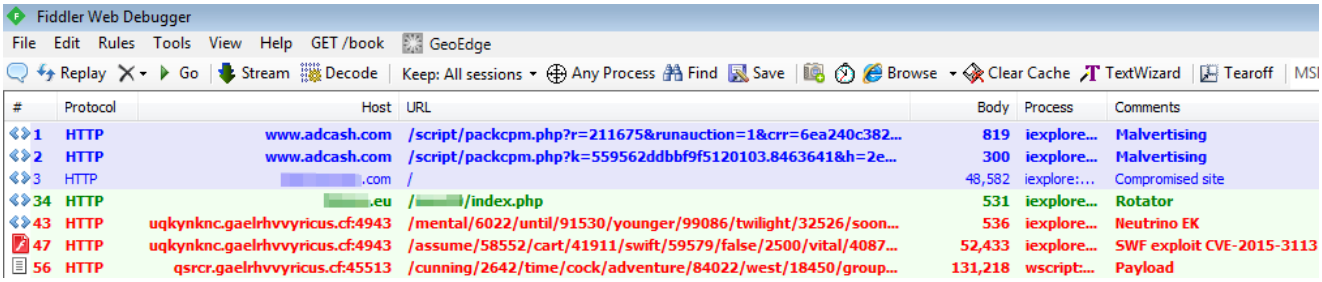
**How is the infection carried?**

Bunitu has been dropped from various exploit kits. On June 10th 2015, as Websense Security Labs described in their post, it was dropped by the Angler Exploit Kit. This time, a similar payload is distributed by Neutrino EK.

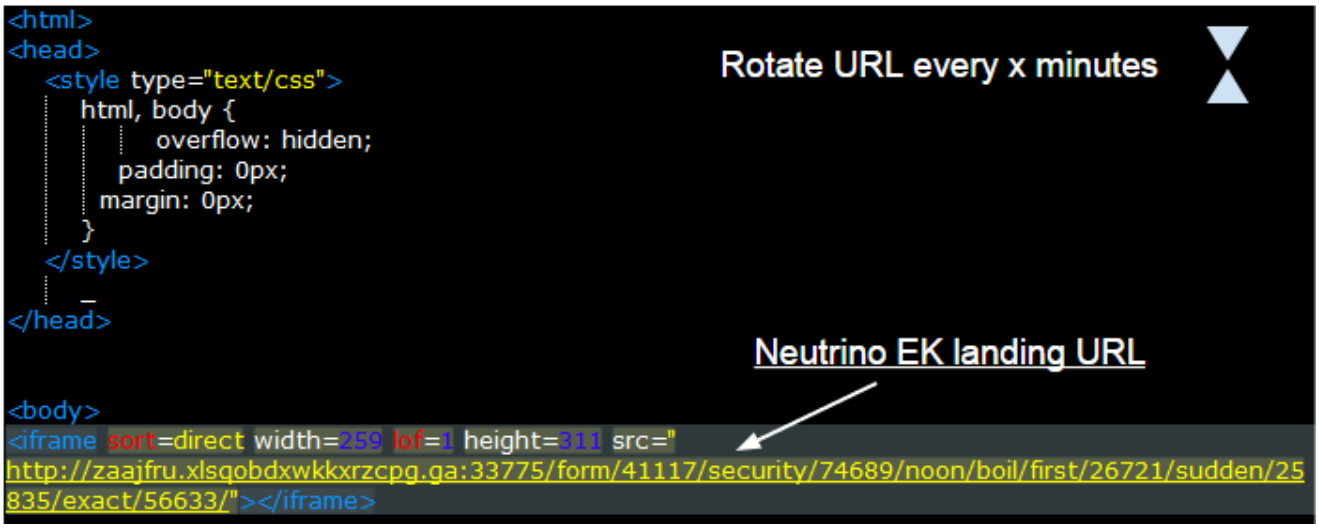
## Role of Neutrino EK

A malvertising from Adcash (they have been notified and the problem is already fixed) redirected users to the Neutrino EK via a compromised site and rotator.

The below screenshot from Fiddler Web Debugger, shows the chain of URLs on the way of dropping the malicious payload:




The rotator (.eu domain) does its job of switching to a new sub-domain every few minutes. This technique is often used to bypass blacklists because the malicious URLs are 'moving targets':



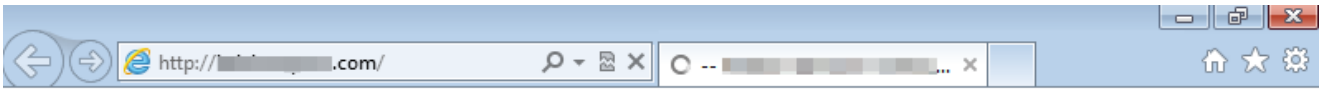
And the landing page carried the exploit:

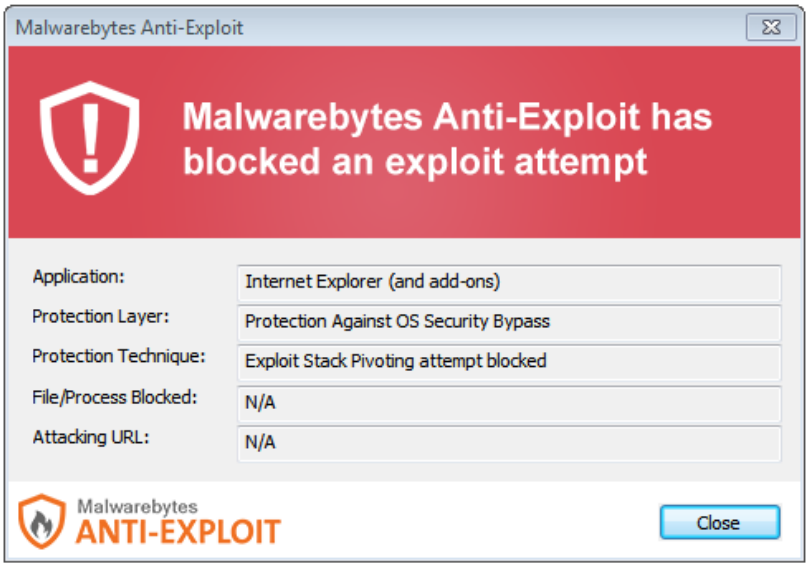
```
<html>
<body>
<script>

</script>
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000" id="ghubj" codebase="
http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=10,1,52,0" width=
"115" height="110">
    <param name="movie" value=
    "/slab.phtml?story=21717&stack=69183&bitter=duchess&endless=hard&boot=98434&moonlight=fifteen&
    expensive=cluster&casual=snore&worth=extreme" />
    <param name="bgcolor" value="#ffffff" />
    <param name="allowScriptAccess" value="always" />
    <embed quality="high" width="115" height="110" src=
    "/slab.phtml?story=21717&stack=69183&bitter=duchess&endless=hard&boot=98434&moonlight=fifteen&
    expensive=cluster&casual=snore&worth=extreme" align="middle" name="ghubj" play="true" loop="false"
    quality="high" allowScriptAccess="sameDomain" type="application/x-shockwave-flash" pluginspage=
    "http://www.macromedia.com/go/getflashplayer"></embed>
</object>

</body>
</html>
```

At this stage, users of Malwarebytes Anti-Exploit were protected – the product detected and stopped the malicious activity.



But if deployed on a vulnerable, unprotected machine, infection followed further – the payload was dropped and deployed.

## Payload: Bunitu Proxy

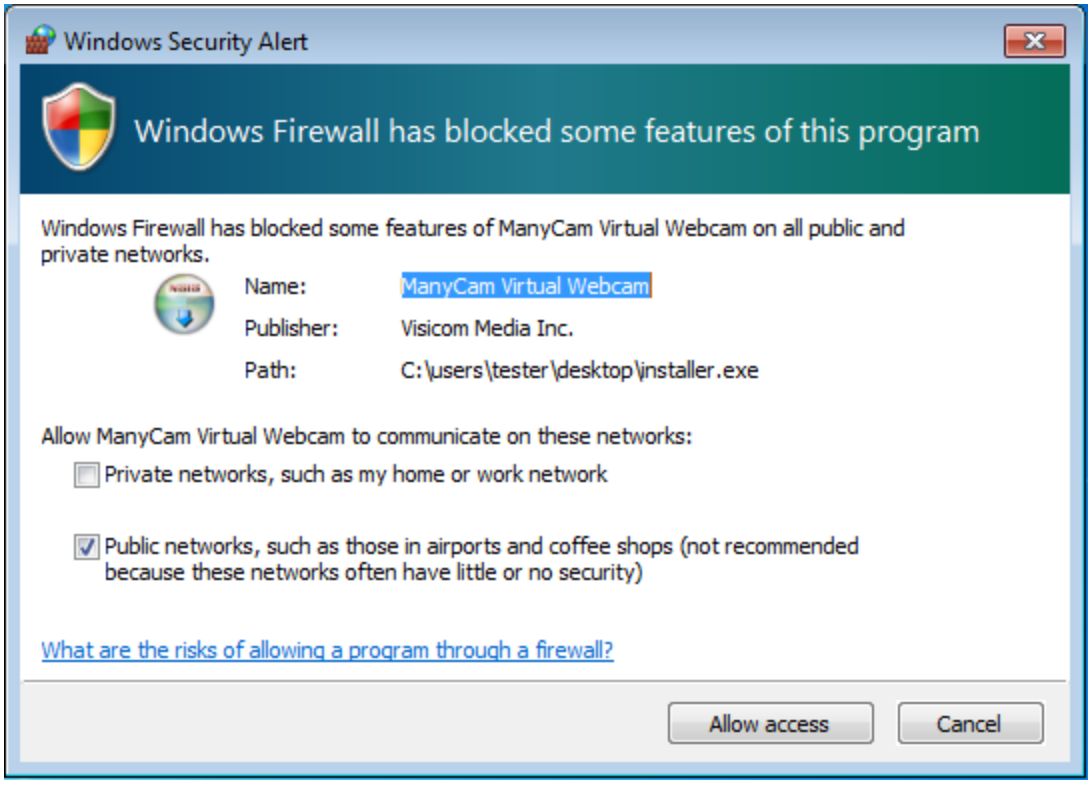
### Infection symptoms

Looking at the payload from outside, we will see just a typical installer (with an NSIS installer icon).

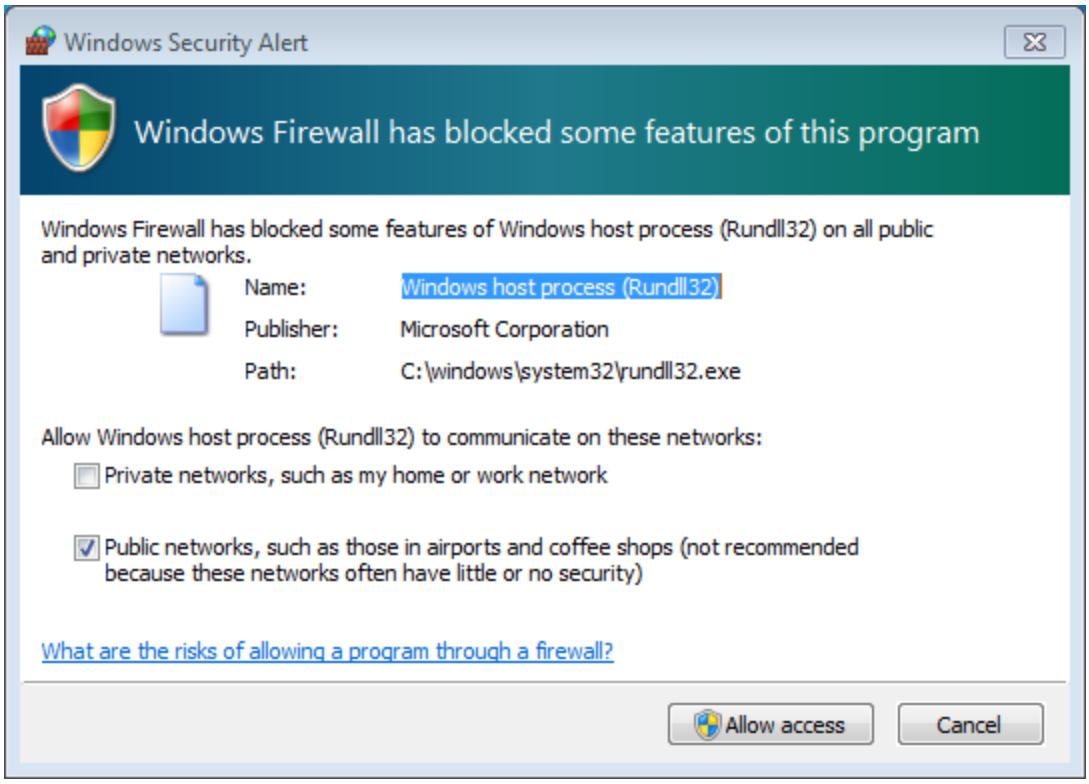
It pretends to be a legitimate piece of software – scamming an existing product: ManyCam by Visicom Media.

After dropping the malicious DLL (described in details further), the installer tries to run it. Then we witness the attempt of opening the ports for incoming connections.

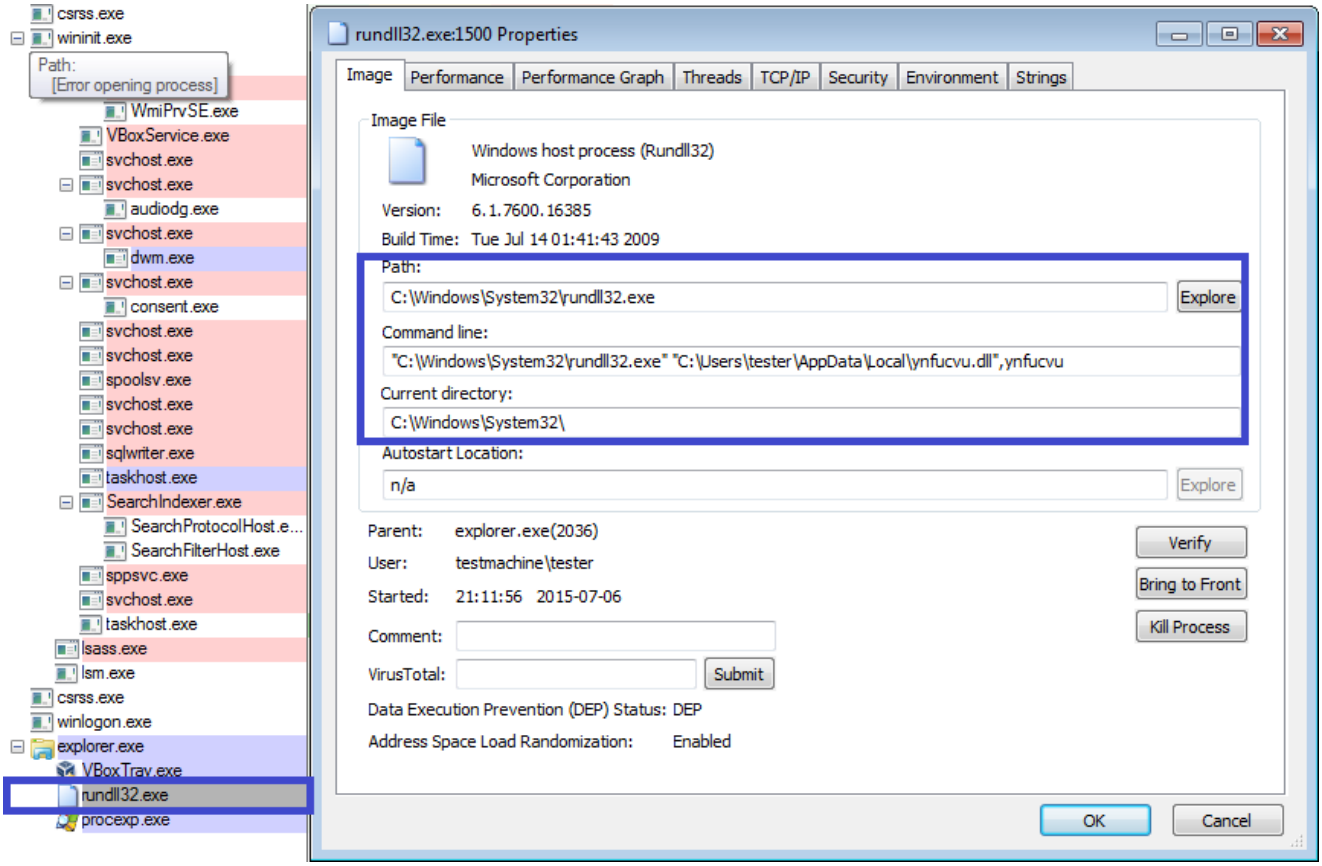
Windows Firewall alerts about this attempt (it seems that at this level it relies on social engineering – only under Windows XP it managed to suppress these messages to maintain stealth).



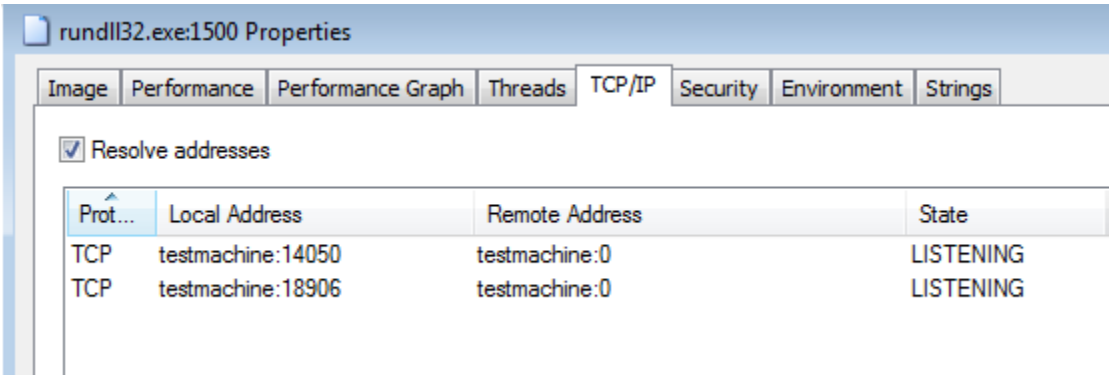

Also, after the successful setup, when the computer is restarted, the persistent module runs again – triggering a similar alert:



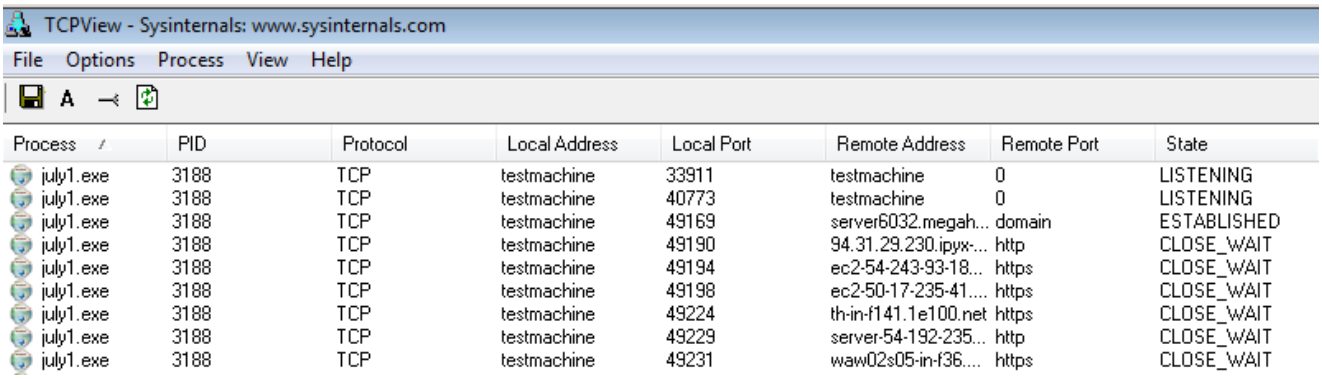

If we see the details of the running process (rundll32) i.e. in Process Explorer, it will reveal the module that has been loaded:



and the open ports (chosen randomly at the time of installation):




If we keep it running for some time, we may even see the clients, that connected via our unwanted proxy (*in the below case, july1.exe was used as the name of the installer*)




## Technical details

To hide its real intentions, the installer uses several layers of protection. It takes several modules to run before the malicious DLL (serving as proxy) is revealed. Let's go deeper!

### Flow:

```
installer.exe-> unpacks and loads:
    lithiasis.dll, function: Avidness -> decrypts and runs using RunPE technique:
        stub_unpacked.exe -> unpacks and loads:
            ynfucvu.dll, function: ynfucvu-> perform all the malicious activities
```

***installer.exe***
Unpacks several files into %APPDATA%/Local/Temp/
*It seems that not all of them play a role in unpacking the payload – some are dropped only to make "noise"*

- [random].tmp , i.e.: nsn4CB0.tmp

- pictures
- script (javascript, YUI module): index(5).php
- **dalookerzmeoajrhja144**
- **UncryptedStub._ini**
- [random].tmp/**lithiasis.dll** (i.e. nse474E.tmp/lithiasis.dll)


Then, it loads the dropped module: **lithiasis.dll** into memory and executes the function called – in the analyzed case – **Avidness** (responsible for further unpacking).

***lithiasis.dll, Avidness***
(real name of the module: __Intelerino.dll)
– is unpacked and loaded by the *installer.exe*
– is obfuscated
– uses files:

- **dalookerzmeoajrhja144** – packed list of functions that are going to be loaded in order to do further unpacking
- **UncryptedStub._ini** – packed executable (I refer to it as: *stub_unpacked.exe*)

Keys used to decrypt the files:

- **dalookerzmeoajrhja144** – “dalookerzmeoajrhja144”
- **UncryptedStub._ini** – “9JKjPZSpEL8uHmkHNlXhwhDc9jRTGN”

Files are encrypted with obfuscated, custom XOR based algorithms. For each file the used algorithm is slightly different. Below you can see sample python scripts for decoding the files: Bunitu Proxy – decoding scripts (github)

**#1 Decrypting functions**


```
def decode1(data, key, max_key):
    l = len(key)
    j = 0 #key index
    decoded = bytearray()
    for i in range(0, len(data)):
        decoded.append(data[i] ^ key[j % l])
        if (i > 0):
            j += 1
        if (j == max_key):
            j = 0
    return decoded
```

## #2 Decrypting PE file

```
1000182B   8B85 CCFEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x134]   decrypt PE
10001831   8B95 14FEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x1EC]
10001837   3BC2               CMP EAX,EDX
10001839   0F8D F2000000      JGE lithiasi.10001931
1000183F   8B85 20FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x1E0]
10001845   8B95 CCFEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x134]
1000184B   0FB60402           MOVZX EAX,BYTE PTR DS:[EDX+EAX]
1000184F   0385 A4FEFFFF      ADD EAX,DWORD PTR SS:[EBP-0x15C]
10001855   8B95 20FEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x1E0]
1000185B   8B8D CCFEFFFF      MOV ECX,DWORD PTR SS:[EBP-0x134]
10001861   880411             MOV BYTE PTR DS:[ECX+EDX],AL
10001864   8B85 20FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x1E0]
1000186A   8B95 CCFEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x134]
10001870   0FB60402           MOVZX EAX,BYTE PTR DS:[EDX+EAX]
10001874   8B95 A4FEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x15C]
1000187A   8B8D A8FEFFFF      MOV ECX,DWORD PTR SS:[EBP-0x158]
10001880   8945 E4            MOV DWORD PTR SS:[EBP-0x1C],EAX
10001883   8BC2               MOV EAX,EDX
10001885   99                 CDQ
10001886   F7F9               IDIV ECX
10001888   8B85 C4FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x13C]
1000188E   0FB60402           MOVZX EAX,BYTE PTR DS:[EDX+EAX]
10001892   8B55 E4            MOV EDX,DWORD PTR SS:[EBP-0x1C]
10001895   33D0               XOR EDX,EAX
10001897   8B85 20FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x1E0]
1000189D   8B8D CCFEFFFF      MOV ECX,DWORD PTR SS:[EBP-0x134]
100018A3   881401             MOV BYTE PTR DS:[ECX+EAX],DL
100018A6   8B85 20FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x1E0]
100018AC   8B95 CCFEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x134]
100018B2   0FB60402           MOVZX EAX,BYTE PTR DS:[EDX+EAX]
100018B6   8B95 CCFEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x134]
100018BC   8B8D A8FEFFFF      MOV ECX,DWORD PTR SS:[EBP-0x158]
100018C2   8945 E8            MOV DWORD PTR SS:[EBP-0x18],EAX
100018C5   8BC2               MOV EAX,EDX
100018C7   99                 CDQ
100018C8   F7F9               IDIV ECX
100018CA   8B85 C4FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x13C]
100018D0   0FB60402           MOVZX EAX,BYTE PTR DS:[EDX+EAX]
100018D4   8B55 E8            MOV EDX,DWORD PTR SS:[EBP-0x18]
100018D7   33D0               XOR EDX,EAX
100018D9   8B85 20FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x1E0]
100018DF   8B8D CCFEFFFF      MOV ECX,DWORD PTR SS:[EBP-0x134]
100018E5   881401             MOV BYTE PTR DS:[ECX+EAX],DL
100018E8   FF85 A4FEFFFF      INC DWORD PTR SS:[EBP-0x15C]
100018EE   8B85 CCFEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x134]
100018F4   8B95 A8FEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x158]
100018FA   8955 EC            MOV DWORD PTR SS:[EBP-0x14],EDX
100018FD   99                 CDQ
100018FE   8B4D EC            MOV ECX,DWORD PTR SS:[EBP-0x14]
10001901   F7F9               IDIV ECX
10001903   8B85 74FEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x18C]
10001909   3BD0               CMP EDX,EAX
1000190B   75 0A              JNZ SHORT lithiasi.10001917
1000190D   C785 A4FEFFFF 0000 MOV DWORD PTR SS:[EBP-0x15C],0x0
10001917   FF85 CCFEFFFF      INC DWORD PTR SS:[EBP-0x134]
1000191D   8B85 CCFEFFFF      MOV EAX,DWORD PTR SS:[EBP-0x134]
10001923   8B95 14FEFFFF      MOV EDX,DWORD PTR SS:[EBP-0x1EC]
10001929   3BC2               CMP EAX,EDX
1000192B   0F8C 0EFFFFFF      JL lithiasi.1000183F
10001931   33C0               XOR EAX,EAX                        decrypted PE
```

result – a new PE file (stub_unpacked.exe):

```
Address   Hex dump                                          ASCII
002ADBD0  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZÉ.♥...♦...  ..
002ADBE0  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  Ş.......@.......
002ADBF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
002ADC00  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ............Ç...
002ADC10  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ♫▼‖♫.┤.=↑$ØL=↑Th
002ADC20  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
002ADC30  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS
002ADC40  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
002ADC50  50 45 00 00 4C 01 04 00 C5 CF 94 55 00 00 00 00  PE..LØ♦.┼ØöU....
002ADC60  00 00 00 00 E0 00 0F 01 0B 01 05 0C 00 7C 00 00  ....ó.*Ø♂Ø♣..¦..
002ADC70  00 C2 01 00 00 00 00 00 AB 1A 00 00 00 10 00 00  .┬Ø.....½→...►..
002ADC80  00 90 00 00 00 00 40 00 00 10 00 00 00 02 00 00  .É....@..►...☻..
002ADC90  04 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00  ♦.......♦.......
002ADCA0  00 60 02 00 00 04 00 00 00 00 00 00 02 00 00 00  .`☻..♦.......☻...
002ADCB0  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ..►..►.....►..►..
002ADCC0  00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ....►...........
002ADCD0  0C A1 00 00 A0 00 00 00 00 50 02 00 B8 04 00 00  .í..á....P☻.Ş♦..
```

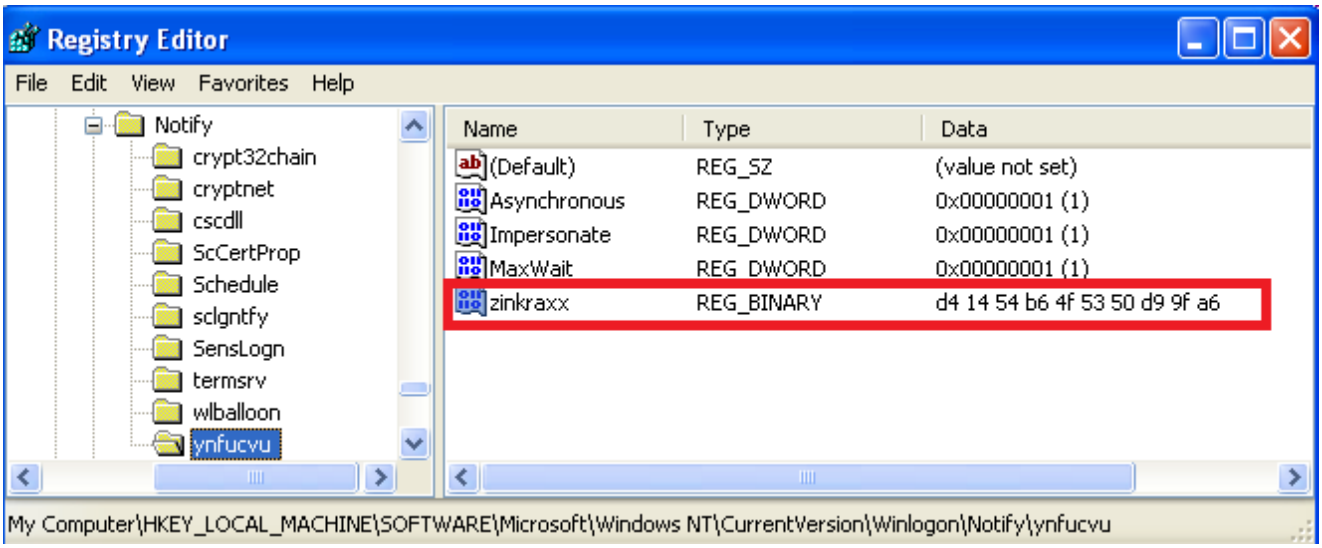
```
def decode2(data, key, max_key):
    j = 0 #key index
    prev_j = 0
    decoded = bytearray()
    for i in range(0, len(data)):
        val = data[i] + prev_j
        val = ((val ^ key[j]) ^ key[prev_j]) % 256
        decoded.append(val)
        prev_j = j
        j = j + 1
        if (j == max_key):
            j = 0
    return decoded
```

After decrypting the new executable: *stub_unpacked.exe* – it loads it into the memory using "RunPE" technique (unmaps the installer.exe and loads the new PE section by section on it's place).

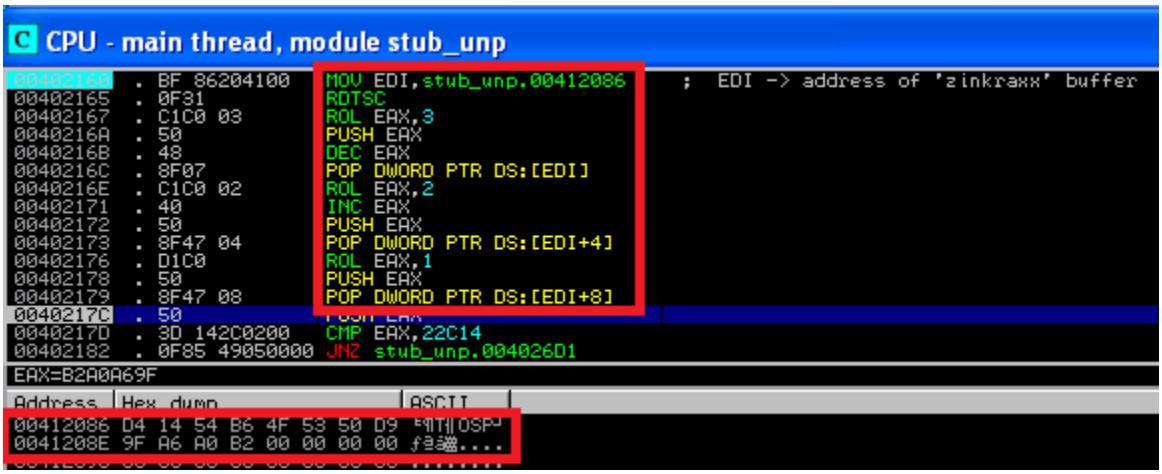
***stub_unpacked.exe***

Its main role is to unpack from inside the "heart" of the malware: module ***ynfucvu.dll***. It also loads and deploys it.

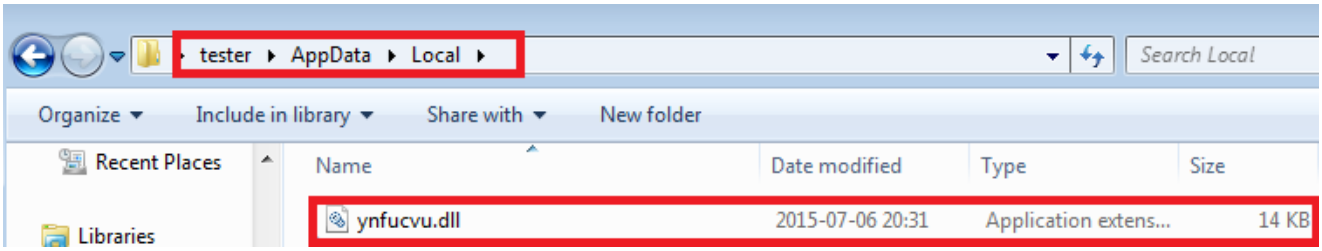
Makes following registry keys (Winlogon Notify):



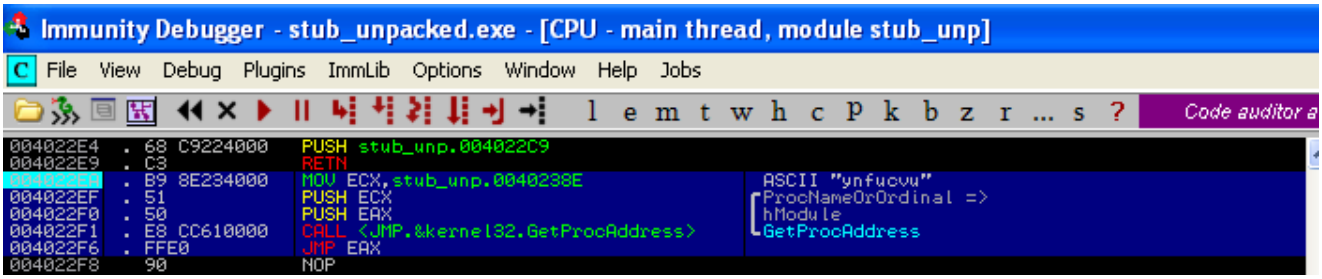

The key 'zinkraxx' is used to uniquely identify the installation. It is made by following simple algorithm:




It uses RDTSC  (an instruction that reads time-stamp counter into EDX:EAX). Then part of the result (EAX) is processed and writen into a buffer. This buffer is then stored in the registry.
After unpacking the DLL it drops it in %APPDATA%/Local folder:



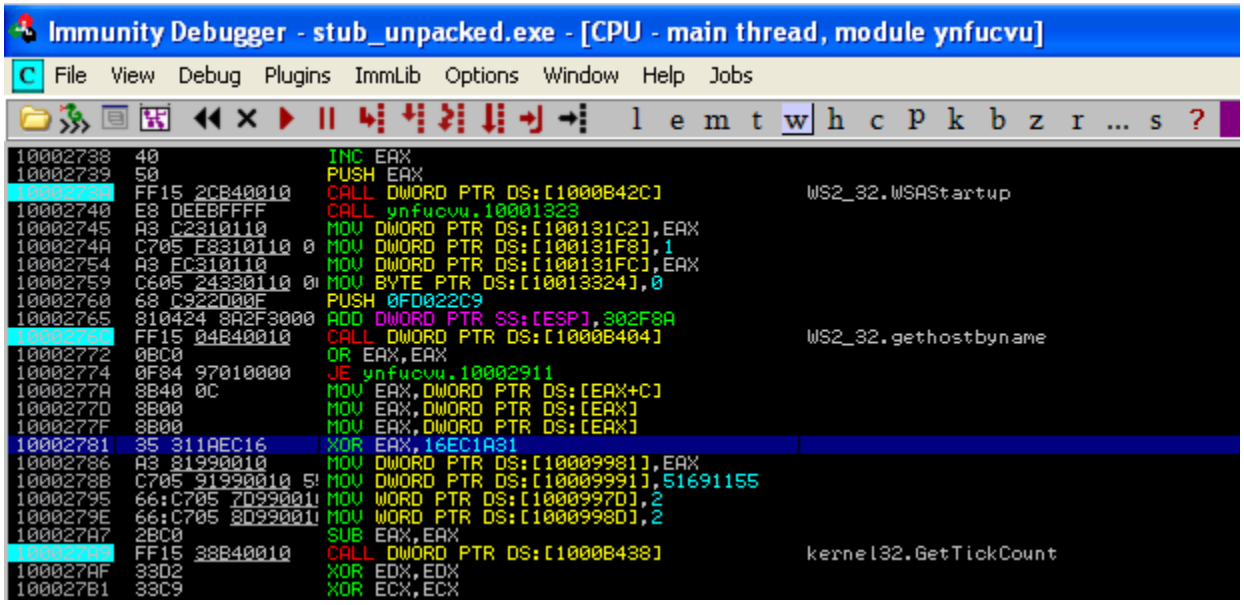

Then, it loads in the memory and enters in the function ***ynfucvu*** of ***ynfucvu.dll*** – using JMP EAX:



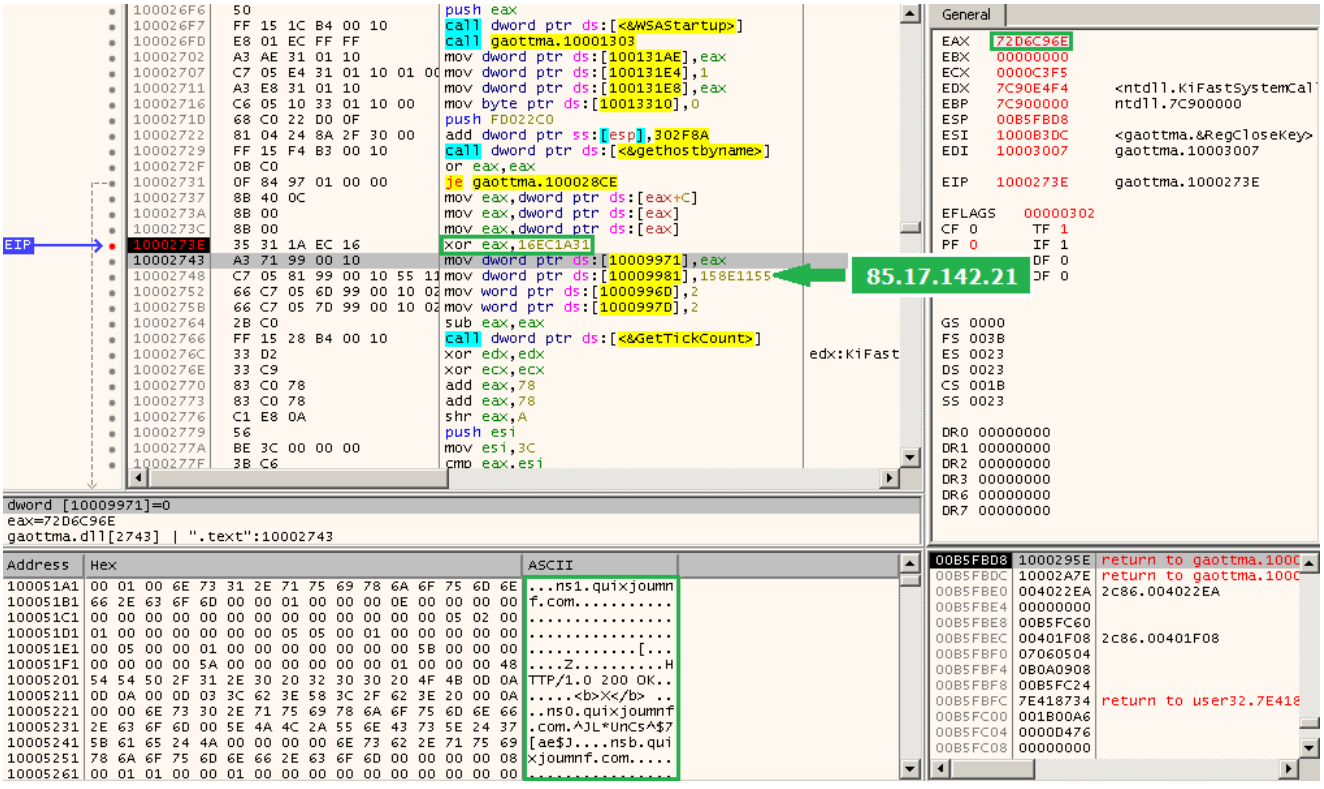

### *ynfucvu.dll, ynfucvu*

This is the Bunitu Proxy module – malicious part of the full package. It is independent from other modules. Once installed, it is loaded on system startup, using rundll32.exe. The entry point is in the function ***ynfucvu***.

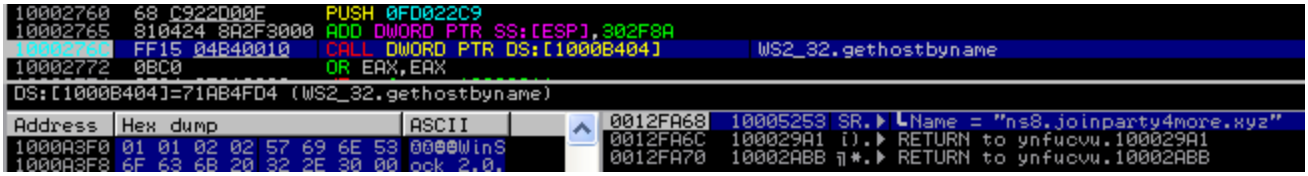
It carries all the network operations – registers the client on the server, opens ports and serves as a proxy.
Techniques used by the Bunitu Proxy module haven't changed much from June 10th, when it was described by Websense Security Lab. Even the xor-ed value is exactly same!




compare with the WebSense analysis:



This module is slightly obfuscated – i.e. domains used to resolve C&Cs are given in a plain text. Only their addresses are calculated on the fly – to make difficult finding where they are referred. As we see below: the address of the string is calculated on the stack (this DLL is always loaded on the same, predefined base – what makes calculation on the addresses easy).




It is also responsible for creating registry keys used for persistence and tries to be invisible for the firewall – by adding itself to the list of Authorized Applications (but effectiveness of it varies depending on the version of Windows).

### Analyzed sample

Original sample (installer) md5=542f7b96990de6cd3b04b599c25ebe57 ; payload (ynfucvu.dll) md5=1bf287bf6cbe4d405983d1431c468de7

## Conclusion

It seems that this malware is being actively distributed through various exploit kits. However, the mutation of the core is not so fast, as we see our sample is very similar to the one observed a month ago. Still, the used packing, composed of many layers gave it advantage of low detection rates in early days after the release.

On the other hand, the good news is that it's not an entirely stealthy piece of malware (except on Windows XP), so a cautious user can notice some of the alarming symptoms.

***Part II: Who's Behind Your Proxy? Uncovering Bunitu's Secrets***