Don't @ Me: URL Obfuscation Through Schema Abuse | Mandiant

By Mandiant

Published: 2023-05-22 · Archived: 2026-04-05 15:39:19 UTC

Written by: Nick Simonian

A technique is being used in the distribution of multiple families of malware that obfuscates the end destination of a URL by abusing the URL schema. Mandiant tracks this adversary methodology as "URL Schema Obfuscation”. The technique could increase the likelihood of a successful phishing attack, and could cause domain extraction errors in logging or security tooling. If a network defense tool is relying on knowing the server a URL is pointing to (e.g. checking if a domain is on a threat intel feed), it could potentially bypass it and cause gaps in visibility and coverage. Common URL parsing logic will fail when encountering this technique, resulting in the loss of visibility into threat campaigns and actor infrastructure.

Network defenders should check if URLs abusing the schema to obfuscate the destination cause any failures in logging, visibility, or security tooling.

Initial Lead

A tweet by @ankit_anubhav was observed describing a technique being used by SMOKELOADER to obfuscate URL destinations. Mandiant’s investigation into this technique discovered multiple other formats of the obfuscation being used to distribute a multitude of malware variants.

In their tweet, they use the URL " `hxxp://google.com@1157586937` ” as an example that ends up opening a Rick Roll video. While the destination is upsetting, this tweet shows two obfuscation techniques being used simultaneously:

- The usage of an "@” sign to obscure the destination server
- The usage of alternative hostname formats to obscure the destination IP address

The "@” Sign

To start, it helps to understand how a URL is structured and parsed by browsers when clicked.

RFC1738 documents the structure for URLs. In section 3.1 (Common Internet Scheme Syntax), it lays out the basic structure for all URLs:

```
<scheme>//<user>:<password>@<host>:<port>/<url-path>
```

In section 3.3 (HTTP), it states that the format for an HTTP URL follows the structure of:

```
http://<host>:<port>/<path>?<searchpart>
```

The RFC specifically states that "No user name or password is allowed.” The user name is defined as the text prior to the "@” sign. When a browser interprets a URL with the username section populated (anything before the "@” sign), it discards it, and sends the request to the server following the "@” sign.

Comparing it with the URL in the tweet shows that the "google.com” section of the URL is being treated as a username. This can be modified as needed to better trick victims to click a link in spear phishing campaigns. For example, it could be replaced with the target email address domain and become much more effective.

Alternative Hostname Formats

In the example, the digits "1157586937" are being treated as the host. However, it's very uncommon for a server IP address to be depicted as an integer. This is the second level of obfuscation.

A dotted-quad IP address is a common representation of an IPv4 address, consisting of four decimal numbers separated by dots, with each decimal number representing 8 bits of the IP address. For example, the IP address 1.2.3.4 can be represented as the binary number 00000001.00000010.00000011.00000100.

This binary representation of an IP address can then be converted to a single integer by treating it as a single, large binary number and converting it to decimal. For example, the binary number 00000001.00000010.00000011.00000100 becomes the decimal number 16909060.

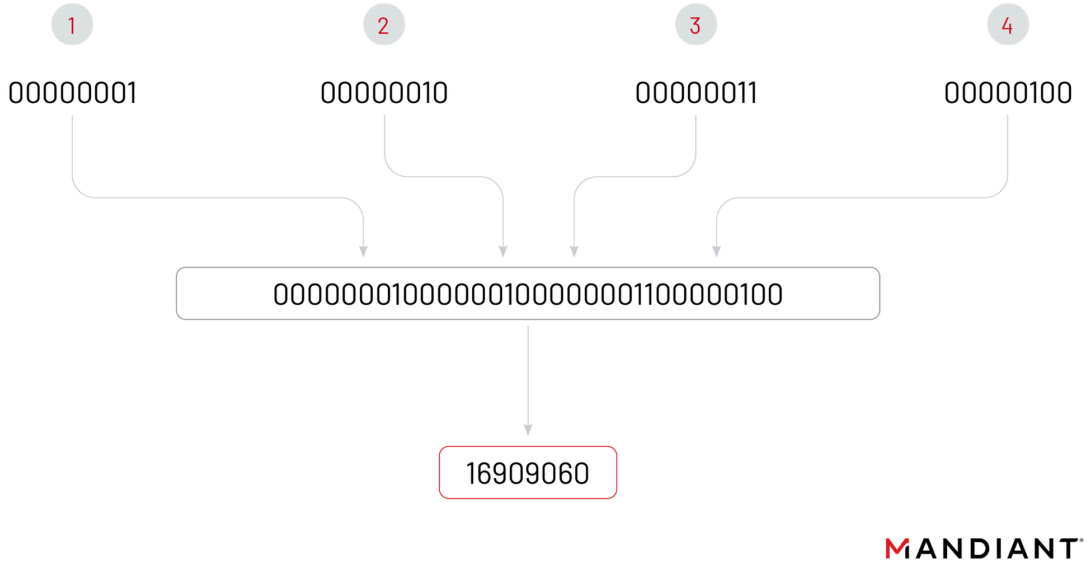

Figure 1: Showing how the IP 1.2.3.4 translates to a decimal representation

Browsers do this conversion automatically. Not only single integer representations can be used:

Hexadecimal can also be put into a dotted-quad format:

```
hxxp://google.com@0xC0.0xA8.0x0.0x1
```

Octal is also possible:

```
hxxp://google.com@0300.0250.0000.0001
```

They can also be mixed to create a truly confusing destination:

```
hxxp://google.com@0xc0.168.0x0.1
```

Domains can also be used, which can be made to look like legitimate destinations:

```
hxxp://legit.banking.site.com@loginportal.onlinebanking.orly/loginPortal.php
```

There are publicly-available tools that can do this level of obfuscation. IPFuscator by Vincent Yiu, for example, generates multiple variations including mixed-type and padded values.

In The Wild

VirusTotal shows usage dating back to at least February 2022. The continuing use of URL Schema Obfuscation is likely because it's working for the attackers, either by decreasing detections by security tooling, or increasing the likelihood a victim clicks the link.

Oftentimes, the URL is used to download additional malware for execution. These have been seen exploiting multiple vulnerabilities to gain code execution on the victim. Most prevalently, CVE-2017-0199 usage has been detected in multiple downloaded documents, along with CVE-2017-11882. A wide range of commodity malware families have been seen using this technique to gain execution, including LOKIBOT, MATIEX, FORMBOOK, and AGENTTESLA.

Example of Recent Abuse

In February 2023, a component file of a Microsoft Word document was discovered using a YARA rule (see Appendix 1) in a VirusTotal Retrohunt. The attack chain had multiple stages, utilizing a template injection attack and an exploit, and dropping AGENTTESLA, which exfiltrated data via an encrypted Telegram channel.

Filename	**PO.docx**
MD5	291f6887bdaf248c7f0cdc9e2c9515cb
SHA-256	7dcbd34116b44f88962e2de72a92849304804fa5141513a35a023f5ab510b3bf

PO.docx was first seen on VirusTotal on February 6, 2023. It contains a template injection technique that runs when the document is opened, requesting the next stage of malware. If the document is decompressed, the next stage of the infection chain can be seen inside the webSettings.xml.rels file:

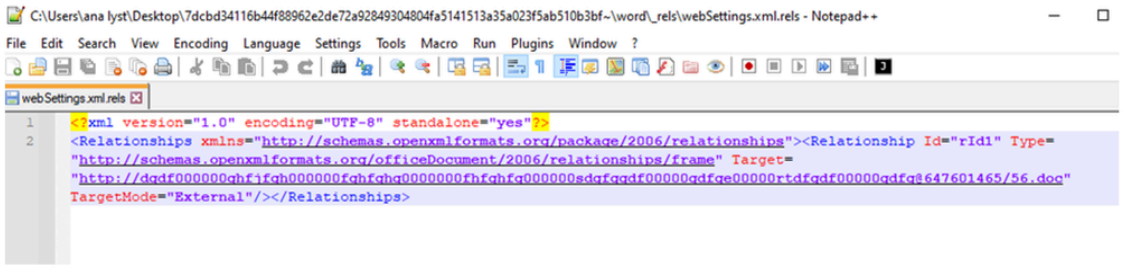

Figure 2: webSettings.xml.rels file

This downloads and opens the obfuscated URL: hxxp://dgdf000000ghfjfgh000000fghfghg0000000fhfghfg000000sdgfggdf00000gdfge00000rtdfgdf00000gdfg@647601465/56.dc

This URL, when deobfuscated, is `hxxp://38.153.157.57/56.doc`. The template injection results in the following network request being made from Microsoft Word. Note the obfuscation isn't present in network traffic; the username field has been stripped out, and the integer representation of the IP address has been changed to the dotted-quad format.

```
GET /56.doc HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/6.0; SLCC2; .NET CLR
2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; .NET4.0E; ms-
office; MSOffice 16)
Accept-Encoding: gzip, deflate
Host: 38.153.157.57
Connection: Keep-Alive
```

Filename	**56.doc**
MD5	fd3ef9f75b0be31f0a482f60a387cb76
SHA-256	1b93a3abb08c33bea46795890d311a201daa56080c4c14eda338eea19a4b4625

The downloaded document, 56.doc, was first seen on VirusTotal on February 6, 2023, the same day as PO.docx. 56.doc is an RTF file that exploits CVE-2017-11882 to download and execute

`hxxp://38.153.157.57/156/vbc.exe`. Unlike PO.docx, this doesn't use the URL obfuscation technique.

Filename	**vbc.exe**
MD5	cea776885d515fe1e88bccb71c016af3
SHA256	d8be588eb6eedc59b033c43150cf324fb8e56050e359b47da8017f4c47d264da

This is internally named "NNbHhH.exe", and is AGENTTESLA, sending stolen data via Telegram with the bot ID and token "`6010275350:AAH4W3CDRhQk0wgfyhQ_jITTy3QYmrxdDbw`"

Detecting

When writing a complex Regular Expression (Regex), it's quite common to see if there's one publicly available that can be reused, instead of starting from scratch and missing potential edge-cases that may not be known. Interestingly, the top voted StackOverflow post for a URL RegEx will fail to find URLs using these techniques. The RegEx shown on the StackOverflow answer is:

```
https?:\/\/(www\.)?{1,256}\.[a-zA-Z0-9()]{1,6}\b(*)
```

Using that Regex in Regex101 shows it misses the obfuscated URL:

If a security, logging, or threat intelligence tool is built using this regular expression, it will be unable to successfully identify or parse out these obfuscated URLs.

For those on defense, network traffic analysis won't show this technique in use. When a browser receives a request to go to a URL using this syntax, it automatically translates it to a valid destination before issuing the request. Therefore, by analyzing network traffic, you wouldn't see an obfuscated URL.

However, using file-based analysis like YARA or AV/EDR can reveal tools using URL schema obfuscation, as can process execution logs. If a program executes something like Powershell's Invoke-WebRequest module pointing to an obfuscated URL, the obfuscated URL will be shown in the logs. As for detecting it in files, YARA rules are included that can find it in Office documents, RTFs, and PDFs.

Conclusion

URL Schema Obfuscation is currently being abused to deliver malware in a variety of ways, from phishing links to template injection. Defenders need to ensure security tooling and logging systems are able to detect, identify, and parse the correct indicators to ensure defenses aren't bypassed by using a format that isn't RFC-compliant. In lieu of other indicators, detection of URL Schema Obfuscation using the provided YARA rules can be a malicious indicator in itself, helping to detect and prevent intrusions.

Acknowledgements

Special thanks to Connor McLaughlin and Jared Wilson for their assistance every step of the way.

Appendix 1: YARA Rules

```
 rule M_Hunting_ObfuscatedURL_DottedQuad
{
 meta:
   author = "Mandiant"
   description = "Finds URL Schema Obfuscation of the format http://loremipsum@1.2.3.4"
 strings:
   $doc = {d0 cf 11 e0}
   $pdf = {25 50 44 46 2D}
   $docx = {50 4b 03 04}
   $rtf = {7b 5c 72 74}
     $url = /https?:\/\/[\w\d\-\_]{1,255}\@\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}(\:\d{1,5})?/ nocase
   condition:
     ($doc at 0 or $docx at 0 or $pdf at 0 or $rtf at 0) and filesize < 3MB and $url
}
```

```
 rule M_Hunting_ObfuscatedURL_Integer
{
 meta:
   author = "Mandiant"
   description = "Finds URL Schema Obfuscation of the format http://loremipsum@16909060"
 strings:
   $doc = {d0 cf 11 e0}
   $pdf = {25 50 44 46 2D}
   $docx = {50 4b 03 04}
   $rtf = {7b 5c 72 74}
     $url = /https?:\/\/[\w\d\-\_]{1,255}\@\d{8,10}(\:\d{1,5})?/ nocase
   condition:
     ($doc at 0 or $docx at 0 or $pdf at 0 or $rtf at 0) and filesize < 3MB and $url
}
```

```
 rule M_Hunting_ObfuscatedURL_DottedQuadHex
{
 meta:
   author = "Mandiant"
   description = "Finds URL Schema Obfuscation of the format http://loremipsum@0x01.0x02.0x03.0x04"
```

```
 strings:
   $doc = {d0 cf 11 e0}
   $pdf = {25 50 44 46 2D}
   $docx = {50 4b 03 04}
   $rtf = {7b 5c 72 74}
     $url = /https?:\/\/[\w\d\-\_]{1,255}\@0x[a-fA-F0-9]{1,2}\.0x[a-fA-F0-9]{1,2}\.0x[a-fA-F0-9]{1,2}\.0x[a-fA-F0-9]{1,2}(
   condition:
     ($doc at 0 or $docx at 0 or $pdf at 0 or $rtf at 0) and filesize < 3MB and $url
}
```

```
 rule M_Hunting_ObfuscatedURL_DottedQuadMix
{
 meta:
   author = "Mandiant"
   description = "Finds URL Schema Obfuscation of the format http://loremipsum@1.2.0x03.0x04"
 strings:
   $doc = {d0 cf 11 e0}
   $pdf = {25 50 44 46 2D}
   $docx = {50 4b 03 04}
   $rtf = {7b 5c 72 74}
     $url = /https?:\/\/[\w\d\-\_]{1,255}\@(0x[a-fA-F0-9]{1,2}|\d{1,3})\.(0x[a-fA-F0-9]{1,2}|\d{1,3})\.(0x[a-fA-F0-9]{1,2}
   condition:
     ($doc at 0 or $docx at 0 or $pdf at 0 or $rtf at 0) and filesize < 3MB and $url
}
```

```
 rule M_Hunting_ObfuscatedURL_Domain
{
 meta:
   author = "Mandiant"
   description = "Finds URL Schema Obfuscation of the format http://loremipsum@mandiant.com"
 strings:
     $doc = {d0 cf 11 e0}
   $pdf = {25 50 44 46 2D}
   $docx = {50 4b 03 04}
   $rtf = {7b 5c 72 74}
   $url = /https?:\/\/[\w\d\-\_]{1,255}\@([\w\d\-]{1,100}\.){1,10}[\w\d\-]{1,20}(\:\d{1,5})?/ nocase
   $exclusions = /https?:\/\/[\w\d\-\_]{1,255}\@(gmail\.com|hotmail\.com|yahoo\.com|outlook\.com|hotmail\.co\.uk|sentry\.i
   condition:
     ($doc at 0 or $docx at 0 or $pdf at 0 or $rtf at 0) and filesize < 3MB and $url and not $exclusions
}
```

Appendix 2: Malware Families By URL

AGENTTESLA	hxxp://OASOSIDFOSWEROEROOWRWERWEREW WW0W83W338W83WOWRWWRWRWRW9W9R9W9R 9WR9W9RW9R9W9R9W9R0WR7RR7W7RW7RRW7R 66WSD6DSD6S6D6DSD66D6S@39209 5676/58.....58.....doc

LOKIBOT	hxxp://xzcbbsjjfhjsdjzazazasvxcvnbbzaszxccvx@392133367/xzswqqazzza_sxcvbnzazazzzzzzza_zxasdazzzasdzczxc/xzzzcv_qazzxcs.doc
FORMBOOK	hxxp://ZZZJOOIOIOSDP99090SDXDdad9SDED99000DF00DF0SDF00DF0XCCXC0V00S0FDS0F0DF00SSZZZZZZZZ0X0C0XCZZXC0X@392117348/22u.22u.22u.doc
MATIEX	hxxp://WEEEERRRRRRRRRRPPPOOOOSSSSSSSOOOOOPPWEEEEEEEOOOOOOOOCCVVVVVVVVVOVVVVVVVVVVVVVVVVOOOOOO@104.168.32.152/O__O.DOC

Posted in

- Threat Intelligence
- Security & Identity

Source: https://www.mandiant.com/resources/blog/url-obfuscation-schema-abuse