Dridex Malicious Document Analysis: Automating the Extraction of Payload URLs | HP Wolf Security

By Patrick Schläpfer

Published: 2021-01-19 · Archived: 2026-04-05 13:23:07 UTC

Introduction

The last three months of 2020 saw a sustained increase in malicious spam distributing Dridex malware. The number of Dridex samples isolated by HP Sure Click more than tripled in Q4 compared to Q3, representing a 239% increase. According to HP Sure Click telemetry, Dridex is currently the second most widely circulating crimeware family behind Emotet. Although originating in 2012 as a banking Trojan, since 2017 Dridex's operators have increasingly shifted their tactics to delivering ransomware.

Dridex's distributors commonly propagate the malware using malicious Office documents (maldocs) that download the Trojan from a remote web server. Interestingly, since mid-2020 a variant of these maldocs started containing hundreds of URLs from which to download the malware. This technique makes the loader more resilient to takedown action by hosting providers and domain registrars. It also increases the likelihood of successfully downloading the payload. Instead of blocking one URL, network security controls such as web proxies would need to block hundreds of URLs to prevent the malware from being downloaded.

Following this change to Dridex's first stage loader last year, we collected samples and analyzed them. This article gives an overview of how the payload URLs are stored and decoded. We also provide a Python script that extracts the URLs to assist security teams in their blocklisting efforts.

Dridex's Excel loader

We focused primarily on the Excel format loader for this analysis because it was by far the most common file type we encountered. Dridex also has a Word dropper, which we've previously written about. As opposed to a dropper, a malware loader contains at least one network destination which is used to download a payload. This allows the loader to be small but necessitates communication over a network to install the malware. The Dridex loader generally uses the approach shown in Figure 1 to download its payload.

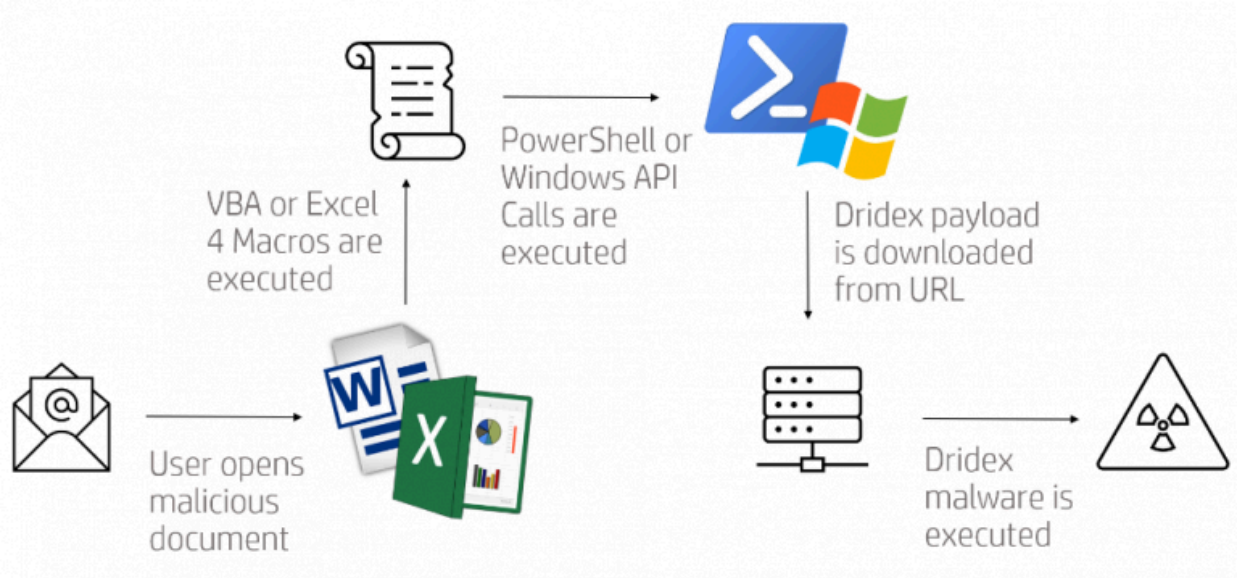


Figure 1 – Typical Dridex infection chain.

Dridex loader types

We identified several types of Dridex loader that each uses a different way to download and execute the payload. Sometimes the document used a Visual Basic for Applications (VBA) macro, Excel 4 macro or both to execute code. The loader uses either PowerShell or Windows API calls to open a network connection and download the payload. We can split Dridex's loaders into two types based on how the code is executed:

Code executed using Excel 4 macros

The first type of loader uses Excel 4 macros to generate PowerShell code or call Windows API functions to download the malware. In both cases, the loader only downloads the payload from one URL.

Code executed using VBA macros

The second type of loader uses VBA macros to download Dridex. This type of loader uses different methods to achieve this. One way is by generating an encoded shell command, which calls PowerShell to download the payload. The loader also only downloads the payload from one URL.

A second, and more notable, method used by this type of loader stores encoded data in an Excel worksheet. When run, the VBA code loads the data from the worksheet and decodes it using one of many routines. The decoded data is a list of hundreds of URLs, from which one is randomly chosen to download Dridex. The sheer quantity of payload URLs is unusual for a loader, so we decided to analyze this download mechanism in more detail.

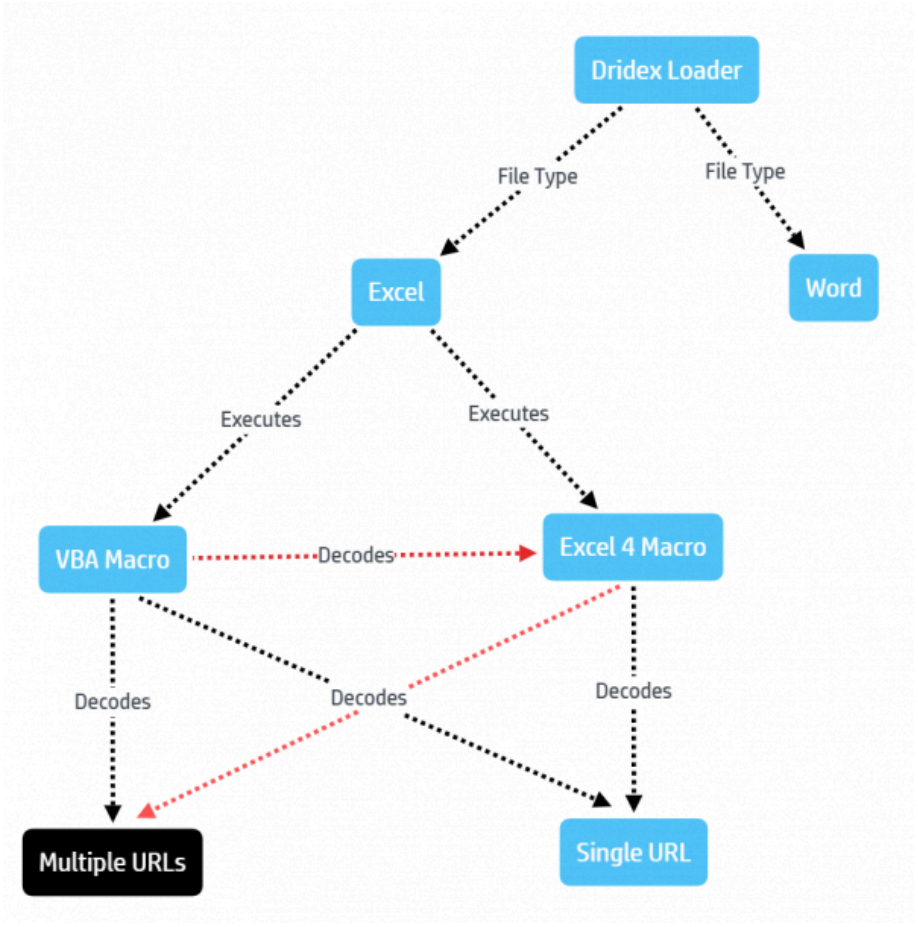


Figure 2 – Overview of Dridex loader types from September to December 2020.

Maldocs containing multiple URLs

The first question we asked about this type of loader was how the URLs are stored in the document. To find this specific answer we analyzed the document by manually debugging the VBA code. Opening the VBA project from a sample in Excel for the first time, generated a warning that the “Project Is Unviewable”. You can learn about the reason for this warning and its solution in [this article](). After resolving this issue, we could read and interact with the loader’s VBA code.

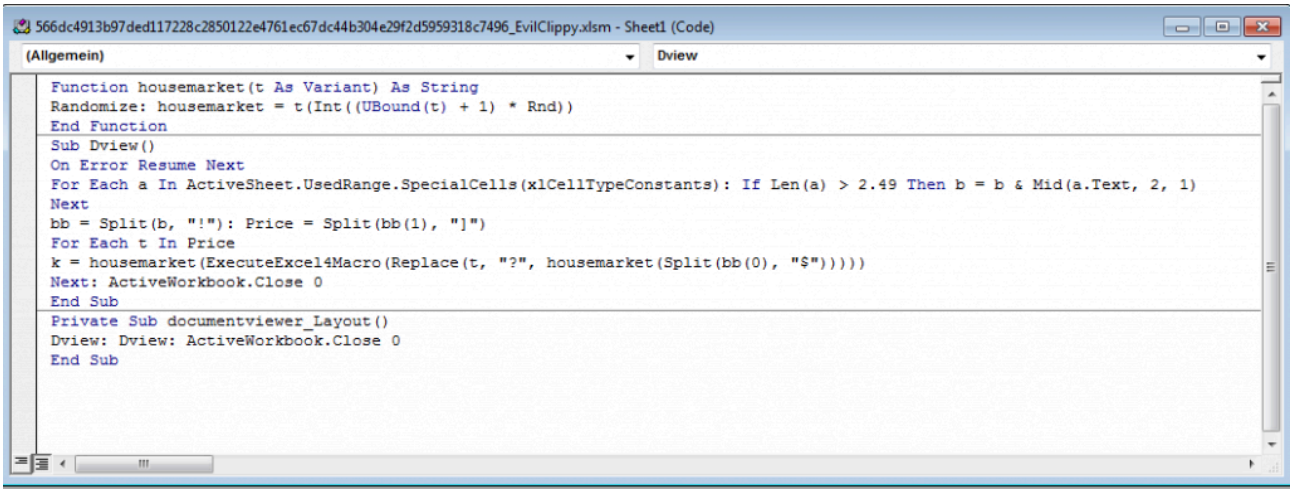
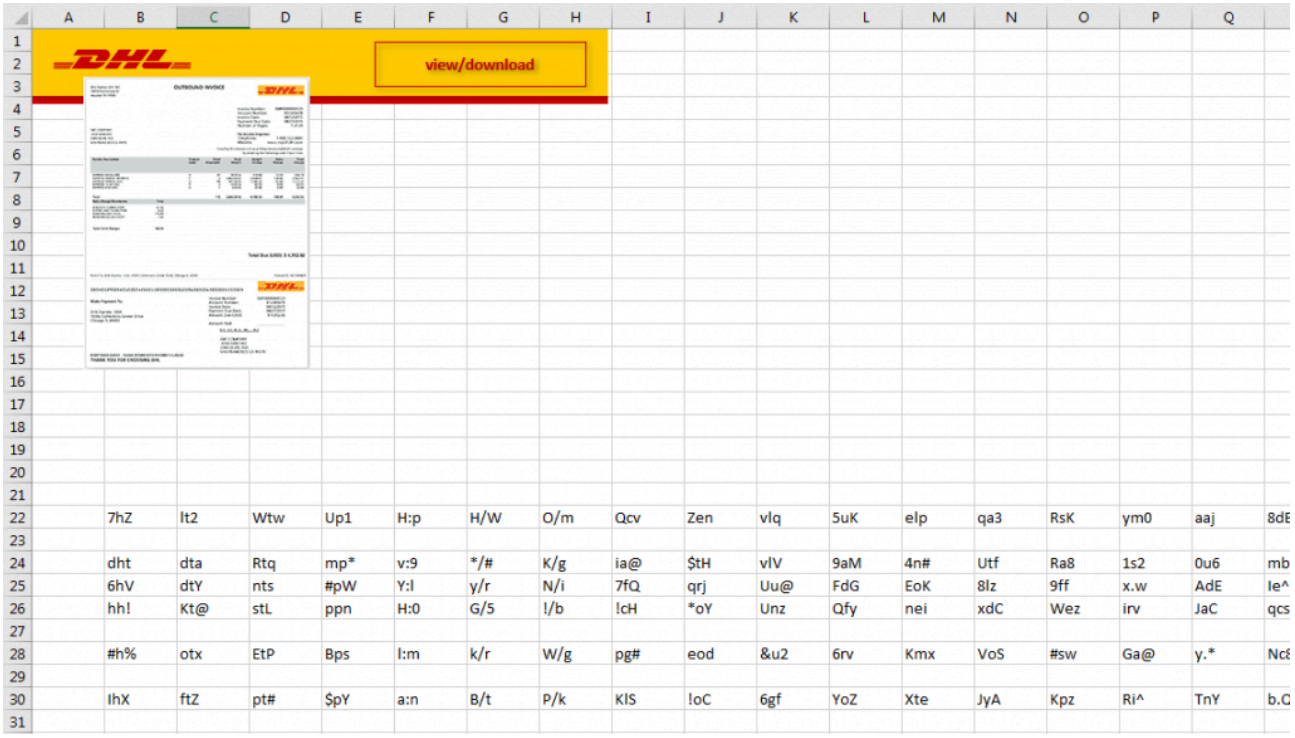


Figure 3 – VBA code of a Dridex loader.

The code executes when the spreadsheet's layout changes, which occurs when the file is opened. The main function contains a loop that iterates over the cells of the active worksheet. During each iteration, encoded data is read from the worksheet, decoded and stored inside a variable. The variable is then split into multiple elements and used as an argument for the *ExecuteExcel4Macro* function. This function, as the name indicates, runs a Microsoft Excel 4 macro given as an argument. The encoded URLs are read from the content of the worksheet. You can see the encoded text by selecting all the cells in the worksheet and changing the text's color.



22	7hZ	lt2	Wtw	Up1	H:p	H/W	O/m	Qcv	Zen	vlq	5uK	elp	qa3	RsK	ym0	aaj	8dE
24	dht	dta	Rtq	mp*	v:9	*/#	K/g	ia@	$tH	vlV	9aM	4n#	Utf	Ra8	1s2	0u6	mb
25	6hV	dtY	nts	#pW	Y:l	y/r	N/i	7fQ	qrj	Uu@	FdG	EoK	8lz	9ff	x.w	AdE	le^
26	hh!	Kt@	stL	ppn	H:0	G/5	!/b	!cH	*oY	Unz	Qfy	nei	xdC	Wez	irv	JaC	qcs
28	#h%	otx	EtP	Bps	l:m	k/r	W/g	pg#	eod	&u2	6rv	Kmx	VoS	#sw	Ga@	y.*	Nc$
30	IhX	ftZ	pt#	$pY	a:n	B/t	P/k	KlS	!oC	6gf	YoZ	Xte	JyA	Kpz	Ri^	TnY	b.C

Figure 4 – Dridex loader worksheet with encoded data.

Since we are interested in the URLs, we slightly modified the VBA code to run the data decoding function and print them out. Figure 5 shows a snippet of these URLs.

```
http://celulasmadreclinicas.com/bsj53ps.pdf
http://atlantasubdivisions.com/prnub4xh.zip
http://frudolf.de/ew1tx34.rar
http://confederaciondetrabajadoresdemexi.co/avq2x9q.zip
http://gourmosa.com/k596w72es.pdf
http://logotypin.ru/g4ea7trqx.zip
http://converafrica.com/oxywy2k8.jpg
http://naon.ru/wxbuqq6.pdf
http://e-kconsulting.co.ke/gv9zl1mso.txt
http://guiaparalapazinterior.com/lj9yxldgh.gif
http://goldengreenindustries.co.za/z4d0a9z01.zip
http://o2executive.com/io7zx7ai.jpg
http://poliss.eu/nemmkm9fl.rar
http://prayonline.it/cm5xssc78.txt
http://munkaerthajduban.hu/1509wm3.rar
http://primal.es/b6egewgab.pdf
http://nouvelle-cosmobel.com/dujxfwhwk.txt
http://psmedien.de/s3j75lm.jpg
http://plasfan.ind.br/g62tkiapp.pdf
http://originador.com/iuwgjwxr.rar
http://rosarybusinessdev.com/n3nbxj41.zip
http://reusite.fr/dxhohc.txt
http://safestflights.com/ofw89uj.zip
```

Figure 5 – Extracted Dridex payload URLs.

Looking at the Excel 4 macro's execution, it is notable that the return value is not evaluated. This means that if the URL host is offline, the loader will not download the payload from another URL.

From analyzing the decoding functions, we found many were re-used or slightly modified between samples. The decoding functions work by iterating over all the values in an active worksheet. In about 60 samples from around 30 Dridex spam waves, we found six types of decoding routines.

1. Character offset encoding

When the worksheet's data is encoded using character offset encoding, the Dridex loader decreases each letter's ASCII value in the URL by one, two or three. How much the ASCII value is decreased is randomly chosen. We can implement a function in Python to decode the string and return the correct URL, as shown below.

```
1  def char_offset_encoding(cell_value):
2      for offset in range(3):
3          potential_url = ""
4          for letter in cell_value:
5              potential_url += chr(int(ord(letter)+offset))
6          if "http" in potential_url:
7              return potential_url
```

Figure 6 – Code snipped from the character offset encoding algorithm.

2. Reverse encoding

A telltale sign of a Dridex loader using reverse encoding is if there are numbers scattered about the worksheet. These numbers indicate the index in an array. The corresponding value is the row number as an ASCII value, which the decoder function converts into a character. The sorted array outputs the list of URLs.

```
reverse_encoding_dict = dict()

def reverse_encoding(cell):
    int_value = int(cell.value)
    reverse_encoding_dict[int_value] = chr(cell.row)
```

Figure 7 – Code snipped from the reverse encoding algorithm implemented in Python.

3. Scramble encoding

Scramble encoding is similar to character offset encoding. The characters are converted from an ASCII value after adding or subtracting another value. The decision about whether a value is added or subtracted depends on the position of the letter.

```
def scramle_encoding(cell_value, offset):
    for i in range(0, len(cell_value)):
        if (i - 1) % 2 == 1:
            potential_url += chr(ord(cell_value[i:i+1])-offset)
        else:
            potential_url += chr(ord(cell_value[i:i+1])+offset)
    return potential_url
```

Figure 8 – Code snipped from the scramble encoding algorithm.

4. Substring concatenation encoding

Documents containing URLs encoded with substring concatenation encoding contain many small strings in the worksheet. The URLs are decoded by iterating over the values. If the value is longer than two characters, then a substring is taken. The substring is appended to a string that contains all the URLs from the document after finishing the iteration.

```
def substring_concat_encoding(cell_value):
    if len(cell_value) > 2:
        urls += cell_value[1:2]
```

Figure 9 – Code snipped from the substring concatenation algorithm.

5. Hexadecimal encoding

Here the characters in the URLs are encoded as hexadecimal values then stored in random cells in the active worksheet. The URLs are decoded by iterating over the values, converting them into characters and appending

them to a string containing all the URLs.

```
def hex_encoding(cell_value):
    hex_encoding_output += chr(int(str(cell_value), 16))
```

Figure 10 – Code snipped from the hexadecimal encoding algorithm.

6. Format encoding

When you open a Dridex loader that uses format encoding, no values are visible in the active worksheet. Instead of cell values, this type of encoding checks if a cell is formatted to decode the URLs. The decoder iterates over all the cells and checks for their format. If a cell is not formatted as “General” then the corresponding column index is taken as an ASCII value, which corresponds
to a URL character.

```
def format_encoding(cell):
    if not cell.number_format == "General":
        format_encoding_output += chr(cell.column)
```

Figure 11 – Code snipped from the format encoding algorithm.

After decoding the contained URLs with one of the explained routines, the Dridex payload is downloaded and the document is automatically closed.

URL encodings timeline

Tracking the usage of the six different encoding mechanisms over time reveals that the distributors of Dridex maldocs tend to switch the type of URL encoding after a few spam waves. Between those changes, the encoding mechanisms largely remained the same with only slight modifications.

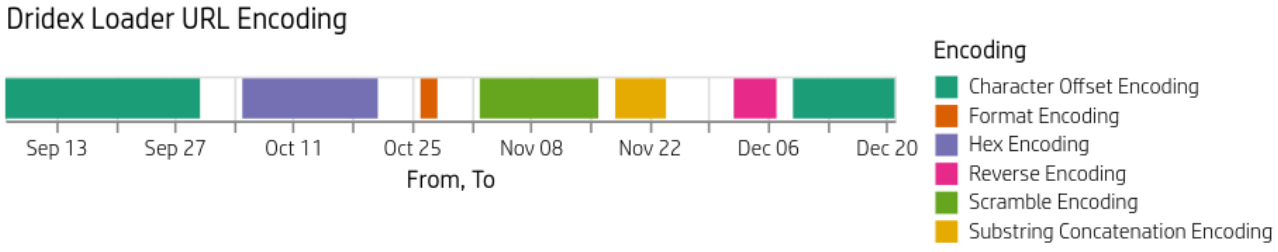


Figure 12 – Types of URL encoding used by the Dridex loader from September to December 2020.

Automatically extracting URLs from Dridex maldocs

We wrote a Python script that extracts all the URLs from Dridex maldocs that use one of the six encoding algorithms. Security teams can use the script’s output to block all potential Dridex payload URLs instead of just one that a typical sandbox would extract through dynamic analysis.

The threat actors behind Dridex and its distribution are continually changing their techniques, so we expect the encoding algorithms to change. Although the automation may not work on future loaders, our analysis should give security teams a clue about how to implement new decoding functions.

Conclusion

Since mid-2020, some Dridex maldocs have contained hundreds of URLs hosting the Trojan. Looking at the documents over this period, we identified six encoding types used to obfuscate the URLs. While the encoding varies between spam wave, the documents' basic structure has not changed significantly. We think it is feasible to implement decoders for future encoding types used in Dridex maldocs building on this approach.

Indicators of Compromise (IOCs)

Using the Python script to automate the URLs' extraction, we collected 2,082 URLs from 56 documents. You can find the IOCs corresponding to this analysis and the URL extraction script in the HP Threat Research GitHub repository.
