Regular Expressions for Beginners: How to Get Started Discovering Sensitive Data

Jeff Melnick (/author/jeff_melnick/)

Published: May 29, 2018

Any data discovery and classification solution heavily relies on regular expressions (sometimes called RegExes, REs or RegEx patterns) to identify sensitive data. But what are RegExes and how can they be used to discover sensitive data? Let's find out.

Regular expressions are a small but highly specialized programming language; they are basically wildcards on steroids. Using this little language, you specify rules that define the strings you want to match. For example, you can define a RegEx that will match email addresses, PII, PHI or credit card numbers.

Regex Components

A RegEx can include literals and metacharacters.

Literals

Any single character, except for those reserved as metacharacters, is already a regular expression itself. For example, **www** is a match for **www.netwrix.com** but **wwz** is not. Note that regular expressions are case sensitive, so **www** will not match **WWW** or **wWw**.

Metacharacters

The following single characters are not interpreted as literals but instead have special meanings:

- **. ^ $ * + ? { } [ ] \ | ( )**

The following table describes how each of these metacharacters functions.

Type	Meta-characters	Description	Examples

Type	Meta-characters	Description	Examples
The dot	**.**	The period means any character.	**net.rix** will match both **www.netwrix.com** and **www.netfrix.com**.
Character class	**[]**	Matches for anything inside the square brackets. The one exception is the ^ character. Inside a class, at the beginning, the ^ means exception from the search. For example **[^n]** will match any character except n; this is called a negated character class. Note that metacharacters (with one exception) are not active inside classes. For example, [net$] will match any of the characters **n**, **e**, **t** or **$** (**$** is a metacharacter, but inside a character class it matches only $). The one exception is the ^ character. Inside a class, at the beginning, the ^ means exception from the search. For example **[^n]** will match any character except **n**; this is called a negated character class.	You can list characters individually; for instance, **net[wrx]** will match **netw** , **netr** and **netx** but not **netz**. Or you can look for a range of characters by giving two characters and separating them by a hyphen; for example, **net[a-z]** will match **neta**, **netw** and **netf** but not **net1**.
Anchors	**^**	Used to match characters at the beginning of a string	**^https** will match **https://netwrix.com** but not **www.netwrix.com** or **http://netwrix.com**
	$	Used to match characters at the end of a string	**com$** will match **www.netwrix.com (https://www.netwrix.com/)** or **telecom** but not **computer**.
Iteration / quantifiers	**?**	Matches the preceding element zero or one time (it will always match if the character was not found). It is great for finding optional characters.	**colou?r** will match both **color** and **colour**.
	*****	Matches the preceding element zero or more times instead of zero or once. It is great for finding optional series of characters.	**ne*t** will match **nt** (zero **e** characters), **net** (one **e**), **neeet** (three **e** characters), and so forth.
	+	Matches the preceding element one or more times. Pay careful attention to the difference between * and +. * matches zero or more times, so whatever's being repeated may not be present at all; + requires at least one occurrence.	**ne+t** will match **net** and **neeet** but not **nt**.
	|	The choice operator matches either the expression before or the expression after the operator.	**net|wrix** will match **net** and **wrix**.
	{}	**{x}** matches if the element that precedes it is found exactly **x** times. **{x,y}** matches if the preceding element is found at least **x** times but not more than y times.	**n{3}** will match **nnn** , **nnnn** and **nnnd** (because they all include **n** three times in a row), but it will not match **nnw**. **9{3}** will match **999**, **1234999124** and **text999text**, but not **84299238**, **9909**, or **page992**. **n{3,5}** will match **nnn**, **nnnn** and **nnnnn**.
Blocking and capturing	**()**	Defines a subexpression that can be recalled later using shorthand: The first subexpression in parentheses can be recalled by **\1**, the second can be recalled by **\2** and so on. Parentheses are normally used either with **|** (the choice operator) inside or with quantifiers on the outside.	**Gr(a|e)y** will match **Gray** or **Grey**. **[0-9]([-])[0-9]\1[0-9]** will match **3-4-2** and **4-6-1**, but not **1-23**, **42-1** or **234**.

Type	Meta-characters	Description	Examples
Escape sequence	\	The metacharacter that follows the slash will be used as a literal. Note that some sequences beginning with \ are not escape sequences. Instead, they represent predefined sets of characters that are often useful, such as the set of digits, the set of letters, or the set of anything that isn’t whitespace. The most popular ones are listed below as “special metacharacters.”	**www\.netwrix\.com** will match **www.netwrix.com** but not **www,netwrix,com**.
Special metacharacters	**\s**	Matches any whitespace character (a space, a tab, a line break or a form feed).	**Netwrix\sAuditor** will match **Netwrix Auditor**, and **Netwrix(tab)Auditor**, but not **Netwrix<5 spaces> Auditor** or **NetwrixAuditor**.
	\S	Matches any non-whitespace character.	**\Snetwrix** will match **Xnetwrix** and **1netwrix**.
	\w	Matches any alphanumeric character.	**\w\w\w** will match **net**, **dfw** and **Netwrix**.
	\W	Matches any non-alphanumeric character.	**netwrix\W** will match **netwrix!** and **netwrix?**.
	\d	Matches any decimal digit.	**Netwrix\d\d** will match **Netwrix80** and **Netwrix90**.
	\D	Matches any non-digit character.	**Netwrix\D** will match **Netwrix)** and **Netwrix-**.
	\a	Matches any single alphabetic character, either capital or lowercase.	**net\arix** will match **netWrix**, **netfrix** and **netarix**.
	\b	Defines a word boundary.	**\brix** will match **rix** and **rixon** but not **netwrix**.
	\B	Defines a non-word boundary.	**\Brix** will match **Netwrix** and **trix** but not **rixon**.

Metacharacter combinations

Now we know almost all the metacharacters and are ready to combine them.

Example: Looking for license plate numbers

Suppose we need to find a license number in the format **aaa-nnnn** — the first three digits must be alphanumeric and the last four must be numeric. The hyphen can be replaced with any character or missing altogether.

The RegEx for this will be:

- **\b[0-9A-Z]{3}([^ 0-9A-Z]|\s)?[0-9]{4}\b**

Let’s dissect this RegEx:

- **\b** requires a word boundary, so matching strings cannot be part of a larger string.
- **[0-9A-Z]{3}** means that the first three characters must be alphanumeric.
- **([^ 0-9A-Z]|\s)?** means the next part of the string must be either a delimiter — a non-alphanumeric character or a whitespace character — or nothing at all.
- **[0-9]{4}** means the next part of the string must be 4 digits.
- **\b** specifies another word boundary.

This RegEx will match the following license numbers: **NT5-6345, GH3 9452, XS83289**

However, it will not match these license numbers: **ZNT49371, HG3-29347, nt4-9371**

Example: Looking for Social Security numbers

Another good example is U.S. Social Security number (SSN), which always takes the form **nnn-nn-nnnn**.

The easiest RegEx is the following:

- **[0-9]{3}-[0-9]{2}-[0-9]{4}**

However, this will generate false positives, since not all numbers that have this form are legitimate SSNs. Moreover, it will miss some actual SSNs, including any that are written without the hyphens. To get more accurate results, we should build more complex one. We know that:

- No digit group can be all zeroes.
- The first block cannot be **666** or **900-999**.
- SSNs can be written with whitespace characters instead of hyphens, or without any delimiters at all.
- If the first block starts with a **7**, it must be followed by a number between **0** and **6** and then any third digit.

Therefore, the advanced RegEx will look like this:

- **\b(?!000|666|9\d{2})([0-8]\d{2}|7([0-6]\d))([-]?|\s{1})(?!00)\d\d\2(?!0000)\d{4}\b**

As before, **\b** at the beginning and end specify a word boundary. Let’s look more deeply at each number block in between.

The first block

- **(?!000|666|9\d{2})** is a negative look-ahead that specifies the number must not begin with **000**, **666**, or **9** followed by any two digits.
- **([0-8]\d{2}** specifies that the string has to start with a digit between **0** and **8** and have two more digits **(0-9)** after it.
- **|7[0-6]\d))** says that it happens to begin with **7**, the next digit must be between **0** and **6**, followed by any digit.
- **([-]?|\s{1})** specifies that after the three digits, there should be either a hyphen, a whitespace character or nothing at all to mark the end of the first block.

The second block

- **(?!00)** is another negative look-ahead that specifies there must not be **00** in the second block.
- **\d\d** specifies that there must be any two digits in the second block.
- **\2** matches the same text as the second capturing group, which is **([-]?|\s{1}),** so it specifies that the second block can end with a hyphen, a whitespace character or no additional character at all.

The third block

- **(?!0000)** is another negative look-ahead that specifies there cannot be four zeroes in the third block.
- **\d{4}** requires any four digits in the third SSN block.

Examples of popular RegExes

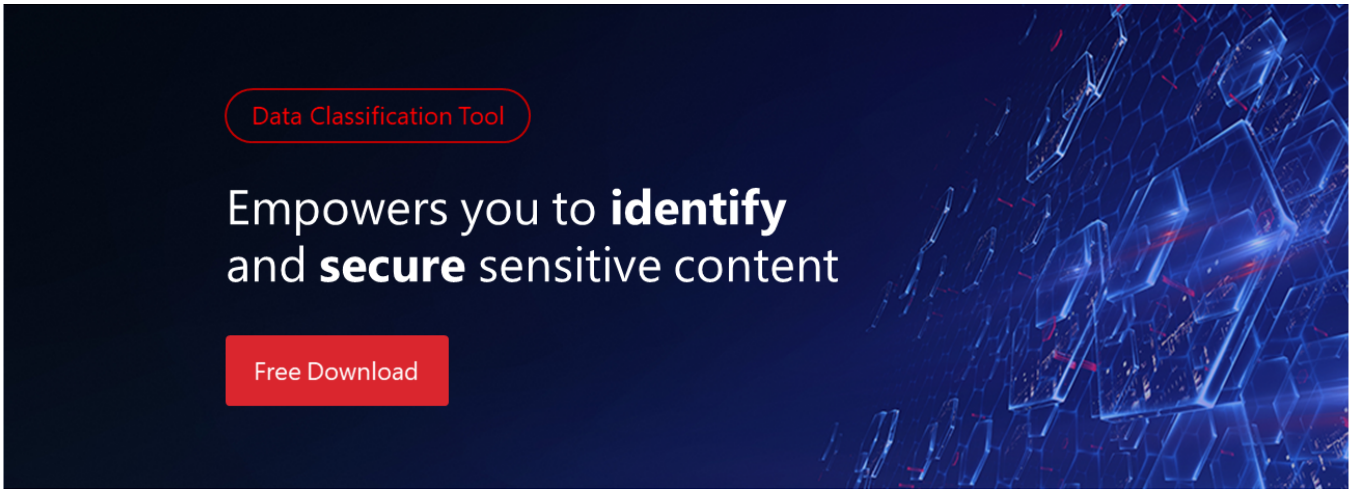
<table>
<tr><th>To find</th><th>Use this RegEx</th><th>Example of match</th></tr>
<tr><td>Email addresses</td><td>^[\w\.=-]+@[\w\.-]+\.[\w]{2,3}$</td><td>T.Simpson@netwrix.com</td></tr>
<tr><td>U.S. Social Security numbers</td><td>\b(?!000|666|9\d{2})([0-8]\d{2}|7([0-6]\d))([-]?|\s{1})(?!00)\d\d\2(?!0000)\d{4}\b</td><td>513-84-7329</td></tr>
<tr><td>IPV4 addresses</td><td>^\d{1,3}[.]\d{1,3}[.]\d{1,3}[.]\d{1,3}$</td><td>192.168.1.1</td></tr>
<tr><td>Dates in MM/DD/YYYY format</td><td>^([1][12]|[0]?[1-9])[\/-]([3][01]|[12]\d|[0]?[1-9])[\/-](\d{4}|\d{2})$</td><td>05/05/2018</td></tr>
<tr><td>MasterCard numbers</td><td>^(?:5[1-5][0-9]{2}|222[1-9]|22[3-9][0-9]|2[3-6][0-9]{2}|27[01][0-9]|2720)[0-9]{12}$</td><td>5258704108753590</td></tr>
</table>

<table>
<tr><th>To find</th><th>Use this RegEx</th><th>Example of match</th></tr>
<tr><td>Visa card numbers</td><td>\b([4]\d{3}[\s]\d{4}[\s]\d{4}[\s]\d{4}|[4]\d{3}[-]\d{4}[-]\d{4}[-]\d{4}|[4]\d{3}[.]\d{4}[.]\d{4}[.]\d{4}|[4]\d{3}\d{4}\d{4}\d{4})\b</td><td>4563-7568-5698-4587</td></tr>
<tr><td>American Express card numbers</td><td>^3[47][0-9]{13}$</td><td>34583547858682157</td></tr>
<tr><td>U.S. ZIP codes</td><td>^((\d{5}-\d{4})|(\d{5})|([A-Z]\d[A-Z]\s\d[A-Z]\d))$</td><td>97589</td></tr>
<tr><td>File paths</td><td>\\[^\\]+$</td><td>\\fs1\shared</td></tr>
<tr><td>URLs</td><td>(?i)\b((?:[a-z][\w-]+:(?:\/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-z]{2,4}\/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:'".,<>?«»“”‘’]))</td><td>www.netwrix.com</td></tr>
</table>

Helpful Regex web resources

- https://regexr.com (https://regexr.com) and https://regex101.com (https://regex101.com) will help you to check your RegExes by highlighting syntax and tooltips.
- https://regexcrossword.com (https://regexcrossword.com) is a crossword puzzle game in which the clues are defined using regular expressions.
- https://www.regular-expressions.info (https://www.regular-expressions.info) a great site with information about regular expressions. In addition, the Notepad++ tool has a RegEx helper extension that will serve you well while you're working with regular expressions.

> **Handpicked related content:**
>
> What Is data classification? (https://blog.netwrix.com/2018/03/15/data-classification-explained-what-it-is-why-you-should-care-and-how-to-perform-it/)
>
> Five reasons to ditch manual data classification methods (https://blog.netwrix.com/2018/05/01/five-reasons-to-ditch-manual-data-classification-methods/)
>
> How to build an effective data classification policy for better information security (https://blog.netwrix.com/2018/05/31/how-to-build-an-effective-data-classification-policy-for-better-information-security/)
>
> 10 ways to improve automated data classification (https://blog.netwrix.com/2018/06/12/10-ways-to-improve-automated-data-classification/)
>
> Microsoft File Classification Infrastructure (FCI) explained (https://blog.netwrix.com/2018/05/22/microsoft-file-classification-infrastructure-fci-explained/)



(https://www.netwrix.com/data_classification_software.html?itm_source=blog&itm_medium=banner&itm_campaign=ddc-release&itm_content=lower-banner)

Data classification (/tag/data-classification/)

Show Comments

Previous post

SysAdmin Magazine: My Precious! - Finding & Securing Sensitive Data (https://blog.netwrix.com/2018/05/24/sysadmin-magazine-my-precious-finding-securing-sensitive-data/)

Next post

How to Build an Effective Data Classification Policy for Better Information Security (https://blog.netwrix.com/2018/05/31/how-to-build-an-effective-data-classification-policy-for-better-information-security/)

POPULAR

1 Windows PowerShell Scripting Tutorial for Beginners (https://blog.netwrix.com/2018/02/21/windows-powershell-scripting-tutorial-for-beginners/)

2 How to Perform IT Risk Assessment (https://blog.netwrix.com/2018/01/16/how-to-perform-it-risk-assessment/)

3 Tutorial: Learn the Basics of Active Directory (https://blog.netwrix.com/2017/04/20/tutorial-learn-the-basics-of-active-directory/)

FEATURED TAGS (/TAGS/)

Active Directory (https://blog.netwrix.com/tag/active-directory/) CISSP (https://blog.netwrix.com/tag/cissp/)

Cyber attack (https://blog.netwrix.com/tag/cyber-attack/) Data classification (https://blog.netwrix.com/tag/data-classification/)

Data governance (https://blog.netwrix.com/tag/data-governance/) Data security (https://blog.netwrix.com/tag/data-security/)

GDPR (https://blog.netwrix.com/tag/gdpr/) Insider threat (https://blog.netwrix.com/tag/insider-threat/)

IT compliance (https://blog.netwrix.com/tag/it-compliance/) IT security (https://blog.netwrix.com/tag/it-security/)

Office 365 (https://blog.netwrix.com/tag/office-365/)

Privileged account management (https://blog.netwrix.com/tag/privileged-account-management/)

Risk assessment (https://blog.netwrix.com/tag/risk-assessment/) SharePoint (https://blog.netwrix.com/tag/sharepoint/)

Windows Server (https://blog.netwrix.com/tag/windows-server/) ... (/tags/)

Stay Connected

About Us

Featured Topics

Free Resources

Network Security

Data Security

Compliance