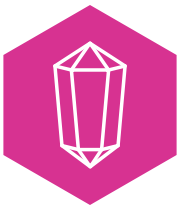
cryptonice Documentation

F5 Labs

Aug 04, 2021

Contents



Making crypto… nice!

Cryptonice is a command-line tool and Python library that allows a user to examine the TLS configuration, certificate information, web application headers and DNS records for one or more supplied domain names. Cryptonice is built heavily upon the excellent SSLyze and Cryptography Python libraries.

Using default arguments Cryptonice will scan the target site on port 443, check for port 80 to 443 redirects, look for DNS CAA records, test the TLS connection and certificate and check for the availability of HTTP/2, HSTS and other security headers.

Currently Supported:

- Windows: Executable, Python scripts and library
- Mac: Python scripts and library
- Ubuntu 20.04: Python scripts and library
- CentOS 8
- Docker container (for Linux)

Sample output of Cryptonice command line usage:

```
cryptonice untrusted-root.badssl.com
```

```
RESULTS
-----------------------------------
Hostname:                         untrusted-root.badssl.com

Selected Cipher Suite:            ECDHE-RSA-AES128-GCM-SHA256
Selected TLS Version:             TLS_1_2

Supported protocols:
TLS 1.2:                          Yes
TLS 1.1:                          Yes
TLS 1.0:                          Yes


CERTIFICATE
Common Name:                      *.badssl.com
Public Key Algorithm:             RSA
Public Key Size:                  2048
Signature Algorithm:              sha256

Certificate is trusted:           False (Mozilla not trusted)
Hostname Validation:              OK - Certificate matches server hostname
Extended Validation:              False
Certificate is in date:           True
Days until expiry:                441
Valid From:                       2019-10-09 23:08:50
Valid Until:                      2021-10-08 23:08:50

OCSP Response:                    Unsuccessful
Must Staple Extension:            False

Subject Alternative Names:
          *.badssl.com
          badssl.com

Vulnerability Tests:
No vulnerability tests were run

HTTP to HTTPS redirect:           True
HTTP Strict Transport Security:   False
HTTP Public Key Pinning:          False

Secure Cookies:                   False

None

RECOMMENDATIONS
-----------------------------------
HIGH - TLSv1.0 Major browsers are disabling TLS 1.0 imminently. Carefully monitor if clients still use this protocol.
HIGH - 3DES The 3DES symmetric cipher is vulnerable to the Sweet32 attack
HIGH - TLSv1.1 Major browsers are disabling this TLS 1.1 immenently. Carefully monitor if clients still use this protocol.
Low - CAA Consider creating DNS CAA records to prevent accidental or malicious certificate issuance.

Scans complete
-----------------------------------
Total run time: 0:00:34.651568
```

For more ideas on how to use Cryptonice view the Examples pages.

CHAPTER 1

Getting help

Need some help? Start here…

- Introducing Cryptonice on F5 Labs
- View the code, fork the project and submit issues on the Cryptonice Github repo.

1.1 Getting Started

1.1.1 Overview

Why Use Cryptonice?

There are many tools available for scanning and analysing the results of TLS and HTTPS configurations. In fact, many of them are web based and do not need to be downloaded. So why use Cryptonice over these services? Cryptonice is aimed at solving these sorts of problems:

Scan multiple sites

Many of the hosted services, such as SSL Labs and Hardenize require you to submit individual sites for sequential scanning. This is ideal for occassional ad-hoc scans, but it's not efficient if you need to scan multiple sites at the same time or continuously scan the same site on a frequent basis. Using either the command line version of Cryptonice, or using its libraries in your own code, you can schedule or programatically scan multiple sites as often as you need to.

Testing internal sites

Externally hosted TLS and security testing sites require that your web app is publicly accessible. For intranet sites or other internal sites, this is of no use. Cryptonice can be run from anywhere and can target internal hostname or IP addresses just as easily as it can scan public facing websites.

Audit the entire HTTPS deployment

A secure HTTPS configuration depends on a lot more than a good certificate and strong ciphers. Cryptonice goes beyond basic TLS scans but also testing your DNS configuration, looking for newer protocols (such as HTTP/2) and also checking for web app headers, such as HTTP strict transport security (HSTS).

Learn how to improve

Cryptonice not only provides a detailed report of the scan results but also shows Critical, High, Medium and low recommendations for how you might be able to make improvements. So you need not be an expert with TLS to understand how to improve the security of your website.

Integrating HTTPS checks in to CI/CD pipeline

More than ever, infrastructure is deployed as code and attackers are equally adept at automating their attacks. Performing automated and regular checks on HTTPS deployments for every code change is essential to ensure accidental configuration changes don't inadvertently weaken the security of the site. By leveraging Cryptonice API's you can build your own checks in to the development workflow.

1.1.2 Install

Docker

Cryptonice is now available as an executable Docker image for Linux systems, meaning it will create the container, run Cryptonice and then close down once it is complete. Using a container avoids the need to install Python, download dependencies or play around with virtual environments. It has been built on a Python 3.7 base image.

Requirements You must already have Docker installed on your system.

Download the Docker container

To grab the latest Docker image:

```
docker pull f5labs/cryptonice
```

To run Cryptonice in the container use the following command (the –rm remove the container once it's completed, and -it makes it interactive so that you can see the output):

```
docker run --rm -it f5labs/cryptonice www.f5.com
```

If you want to output the JSON results to a folder on your local machine you must tell Docker to map a local path to a path within the container. The following example maps a folder in C:Scratch to a new folder called /results within the container):

```
docker run --rm -it --volume //c/scratch:/results f5labs/cryptonice www.f5.com --json_
↪out --json_path /results
```

Once the scan is complete you should find the resulting www.f5.com.json file in your C:Scratch folder.

Windows

Requirements Cryptonice for Windows depends on Visual C++ Redistributable for Visual Studio 2015. Most people are likely to have this installed but if you receive an error about missing file **vcruntime140.dll** then make sure this is installed first.

Standalone executable

If you do not have Python installed, and can't or won't install it, you may be able to use the standalone Windows executable.

1. Head over to the Cryptonice Releases section of the Github repo and download the latest version you find.
2. Once the file has downloaded open it to begin installation (we recommend you install in to a folder that your user has write access to, e.g. C:/Cryptonice)

3. After installation, open a terminal window and navigate in to the installation folder and the version folder found within (e.g. c:/Cryptonicecryptonice-1.0.6)

You may now use Cryptonice by entering the name of the executable and any parameters you need. For example:

```
c:\cryptonice\cryptonice-1.0.6\cryptonice.exe example.com
```

In order to run Cryptonice from any directory you may want to add the installation location to your path, for example:

```
set path=%path%;c:\cryptonice\cryptonice-1.0.6
```

Python app and Library

Installing Cryptonice via pip ensures that you can easily update the tool whenever new versions are released. The other advantage is that Cryptonice should be available to be executed from any directory that you are currently in without modifying your path.

For those that don't yet have Python 3 installed, follow these simple steps.

1. Download Python 3.7 or later (select *Windows x86-64 executable installer*)
2. It is recommended that you leave installation options on default but (optionally) also select... * Select Install for All Users * Select Precompile standard library
3. Open a command prompt and type 'python'. This will send you in to a Python interpreter and also display the version you have installed. This should be 3.7 or later.

Now issue the 'pip' command to install Cryptonice:

```
pip install cryptonice
```

Mac

Python app and Library

Mac OS comes with Python 2 pre-installed. Since Cryptonice requires Python 3.7 or later and you will need to ensure that this is installed.

1. Download Python 3.7 or later (select *macOS 64-bit installer*)
2. Open the downloaded file and follow the installation prompts
3. The install will pop open a file window. Double-click the file *Install Certificates.command* in order to install default root certificates
4. Open up a new terminal window so that we can install Cryptonice

Finally, make sure to use 'pip3' so that the Python 3 version is used. If you issue the 'pip' command then Python 2 will be used and Cryptonice will fail:

```
pip3 install cryptonice
```

Ubuntu

Cryptonice currently supports Ubuntu 20.04. Since this distribution comes with Python 3.8.2 preinstalled we need only install PIP and one dependency manually:

```
sudo apt install python3-pip
sudo apt install python3-pycurl
pip3 install cryptonice
```

CentOS

With a little hand-holding Cryptonice will work on CentOS. It just requires a few manual steps. Currently, CentOS 8 is confirmed to be working. CentOS 8 comes bundled with Python 3.6.8 but we require 3.7 or newer.

Prerequisites

First off, we need to install required development libraries for Python:

```
sudo dnf install gcc openssl-devel bzip2-devel libffi-devel
```

Download Python 3.8

Once that's complete, we'll now download Python 3.8.5 (or any version of Python 3.7 or newer) to your machine:

```
cd /opt
wget https://www.python.org/ftp/python/3.8.5/Python-3.8.5.tgz
```

Extract it with the following command:

```
tar xzf Python-3.8.5.tgz
```

Building and installing

We're now going to configure the source files based on your system and then compile Python on using the altinstall method so that we don't overwrite your current version of Python:

```
cd Python-3.8.5
sudo ./configure --enable-optimizations
sudo make altinstall
```

You can now test that Python 3.8 is installed and working by issuing the following command:

```
python3.8 -V

# Python 3.8.5
```

Installing libcurl*

Installation of Cryptonice will still fail as it cannot find the required version of *libcurl*. Use **yum** to install libcurl:

```
sudo yum install libcurl-devel
```

Installing Cryptonice

Now we can finally use Python 3.8 to install Cryptonice:

```
pip3.8 install cryptonice
```

1.1.3 Command Line

The default command for cryptonice is

```
cryptonice <domain_name>
```

This results in the following commands being run

```
{
    "id": "default",
    "port": 443,
    "scans": ["TLS", "HTTP", "HTTP2", "DNS"],
    "tls_params": ["certificate_information", "ssl_2_0_cipher_suites", "ssl_3_0_
↪cipher_suites","tls_1_0_cipher_suites", "tls_1_1_cipher_suites", "tls_1_2_cipher_
↪suites","tls_1_3_cipher_suites", "http_headers"],
    "http_body": false,
    "force_redirect": true,
    "print_out": true,
    "generate_json": true,
    "targets": [<domain_name>]
}
```

The user can choose to specify custom commands. Each custom command must be preceded with the name of the option (ex: to specify the scans TLS and HTTP to run, the user must add –scans TLS HTTP to the command line parameters)

- –PORT: port to perform the scan on (default = 443)
- –SCANS: scans to perform
  - TLS scan, HTTP headers, HTTP2 check, DNS data (default = None)
- –TLS_PARAMETERS: TLS specific scans to perform (should be listed as specified below, with no commas between options):
  - all, no_vuln_tests, certificate_info, ssl_2_0_cipher_suites, ssl_3_0_cipher_suites, tls_1_0_cipher_suites, tls_1_1_cipher_suites, tls_1_2_cipher_suites, tls_1_3_cipher_suites, tls_compression, tls_1_3_early_data, openssl_ccs_injection, heartbleed, robot, tls_fallback_scsv, session_renegotiation, session_resumption, session_resumption_rate, http_headers
  - all results in all commands being run, no_vuln_tests results in certificate_info, http_headers and the cipher_suites commands being run.
  - More information on each of these scan options can be found at: https://nabla-c0d3.github.io/sslyze/documentation/available-scan-commands.html
- –HTTP_BODY: Y/y or N/n - sets a Boolean variable to include or exclude HTTP pages information
- –FORCE_REDIRECTS: Y/y or N/n - sets a Boolean variable to check for automatic redirects from port 80 to 443 in a TLS scan (default = Y)
- –PRINT_OUT: Y/y or N/n - sets a Boolean variable to print scan results to console (default = Y)
- –JSON_OUT: Y/y or N/n - sets a Boolean variable to print scan results to JSON output file (default = Y)

1.1.4 Examples

Expired Certificates

A simple use for Cryptonice is to check for expiring or expired certificates. This can either be accomplished by using default parameters or specifically (and only) scanning for the website certificate.

Using Cryptonice to check for expired certificates at expired.badssl.com:

```
cryptonice expired.badssl.com --scans tls --tls_parameters certificate_info
```

Results:

```
RESULTS
-----------------------------------
Hostname:                         expired.badssl.com

CERTIFICATE
Common Name:                      *.badssl.com
Public Key Algorithm:             RSA
Public Key Size:                  2048
Signature Algorithm:              sha256

Certificate is trusted:           False (Mozilla not trusted)
Hostname Validation:              OK - Certificate matches server hostname
Extended Validation:              False
Certificate is in date:           False
Days until expiry:                -1923
Valid From:                       2015-04-09 00:00:00
Valid Until:                      2015-04-12 23:59:59

OCSP Response:                    Unsuccessful
Must Staple Extension:            False

Subject Alternative Names:
          *.badssl.com
          badssl.com

RECOMMENDATIONS
-----------------------------------
CRITICAL - Cert Expiry Certificate has expired!

Scans complete
-----------------------------------
Total run time: 0:00:19.095761

Outputting data to ./expired.badssl.com.json
```

We can see from the output, above, that the certificate for expired.badssl.com has expired 1923 days ago and that the certificate is also, therefore, not trusted by the Mozilla root store.

Weak Protocols and Ciphers

Legacy protocols are either deliberately made available in order to accept connections from old browsers or, all too often, they are forgotten about and not removed despite known vulnerabilities.

Cryptonice will detect which ciphersuites are available over which protocol and display warnings should legacy ciphers be found.

Using Cryptonice to check for weak protocols and ciphers on rc4-md5.badssl.com:

```
cryptonice f5.com --scans tls
```

Results:

```
RESULTS
---------------------------------------
Hostname:                          rc4-md5.badssl.com

Selected Cipher Suite:             RC4-MD5
Selected TLS Version:              TLS_1_1

Supported protocols:
TLS 1.2:                           Yes
TLS 1.1:                           Yes
TLS 1.0:                           Yes


RECOMMENDATIONS
---------------------------------------
HIGH - TLSv1.0 Major browsers are disabling TLS 1.0 imminently. Carefully monitor if␣
↪clients still use this protocol.
HIGH - RC4 The RC4 symmetric cipher is considered weak and should not be used
HIGH - MD5 The MD5 message authentication code is considered weak and should not be␣
↪used
HIGH - TLSv1.1 Major browsers are disabling this TLS 1.1 immenently. Carefully␣
↪monitor if clients still use this protocol.

Scans complete
---------------------------------------
Total run time: 0:00:26.915821
```

Overview Understand what Cryptonice is and how it can help you.

Install Get Cryptonice installed on your computer.

Command Line Learn the different command line parameters to customize your scan.

Examples See some examples of basic command line usage of Cryptonice.

1.2 Advanced API Use

1.2.1 Advanced

What is the Cryptonice library?

Installing the Cryptonice library

1.2.2 Modules

scanner.py

def scanner_driver(input_data)

Scanner_driver is the main function of the library from which all other modules can be accessed. It will call functions to collect the requested data, based on the input provided in the input_data dictionary. As results are collected from each module, scanner_driver builds an output dictionary called scan_data with the information. This dictionary is then used to print output to the console, written to a JSON file, and returned to the function that called scanner driver (i.e. a function in another project or from a separate __main__ file).

- *input_data*: dictionary formatted with all necessary scan information (see documentation for example)
- *return*: dictionary of scan data, hostname

def print_output_to_console(scan_data, b_httptohttps)

Print_output_to_console is the function used to print the scan results to the console. This is a useful function to make the data more readable, especially because the JSON output (if created), is written on a single line. Print_output_to_console also generates recommendations to improve the TLS configuration of the target host.

- *scan_data*: dictionary of all output from each module called from scanner_driver
- *b_httptohttps*: Boolean variable noting automatic redirect to HTTPS from HTTP (collected from gethttp.get_http)
- *return*: None

def writeToJSONFile(filename, data)

WriteToJSONFile receives a filename and a dictionary of scan data, and writes it to a single-line JSON file in the same directory as the scanner.py function. If any of the data is not immediately JSON serializable, WriteToJSONFile will call a helper function called print_errors that will attempt to return a string representation of the error. If this fails, it will return a message saying the particular input was not JSON serializable.

- *filename*: string of hostname
- *data*: dictionary of all scan data to convert to JSON
- *return*: None

checkport.py

def port_open(hostname, port)

Port_open is a quick way to check that the port the user would like to scan is open. If it isn't open for connections, the program will terminate early. The function also checks to see if the hostname can also make a TLS connection.

- *hostname*: string
- *port*: int
- *return*: Boolean open_port, open_tls (true if provided port is open, true if TLS is available aka port 443 is open)

modules/gettls.py

def tls_scan(ip_address, str_host, commands_to_run, port_to_scan)

Tls_scan is the main function of the TLS module. A server connection is created and scan commands are queued for the connection. The data from each of the scan commands is appended to a dictionary object which is returned at the end of the function. Certificate information is collected first, including various leaf certificate validity checks, and trust store information. Cryptonice currently only trusts Mozilla certificates, information that is recorded in the cert_errors portion of the tls_scan dictionary data. Cipher suite information is collected, if requested, and the accepted cipher suites for the selected SSL/TLS versions are listed. The vulnerability tests for TLS configuration are recorded largely as boolean variables (true if vulnerable, false if not), except for the ROBOT tests which provides a message related to why it is or isn't vulnerable to ROBOT. After the tests are completed, the data is added to the larger tls_scan dictionary. Metadata is appended and it is returned to the function that called the tls_scan function.

- *ip_address*: string host IP address to scan
- *str_host*: string hostname (SNI)
- *commands_to_run*: list of scan parameters

- *port_to_scan*: int port to scan
- *return*: dictionary of TLS data

def createServerConnection(ip_address, str_host, servers_to_scan, port_to_scan)

CreateServerConnection sets up a connection to the host server. The server connection information is stored in a list called servers_to_scan, which is accessible from the main tls_scan function. The function will return a string error message if the connection failed.

- *ip_address*: string IP address to scan
- *str_host*: string hostname (SNI)
- *servers_to_scan*: empty list to contain server connection information
- *port_to_scan*: int port to scan
- *return*: string "success" or error message

def addScanRequests(scanner, servers_to_scan, commands)

AddScanRequests queues and runs the requested scan commands. This step can take a little while, especially if vulnerability tests are included. The scanner object is updated to hold the results of the scan requests.

- *scanner*: object of sslyze Scanner() class
- *servers_to_scan*: list of open connections
- *commands*: list of string commands
- *return*: None

def getCertificateResults(certificate)

GetCertificateResults is a helper function dedicated to building the dictionary for the certificate data. Each certificate in the deployment chain (usually 2) is sent to this function and parsed for serial numbers, public key, validity dates, signature hash algorithms, and subject alternative names. The

- *certificate*: string literal certificate in PEM format
- *return*: dictionary of certificate data

modules/getdns.py

def get_dns(hostname, all_checks)

Get_dns performs a quick DNS check on the specified hostname. The 'A' records are always collected because the IP addresses for the hostname are stored there. If all_checks is set to true, records for CAA, TXT and MX records will also be collected by the getDNSRecord helper function. The dictionary of DNS data is returned.

- *hostname*: string hostname
- *all_checks*: Boolean variable, true if all DNS checks should be performed, false if they should not be
- *return*: dictionary of DNS data

def getDNSRecord(hostname, record_type)

GetDNSRecord specifically collects the DNS records for a specified record type using the dns.resolver Python library. The list of collected records is returned.

- *hostname*: string hostname
- *record_type*: string record type (A, CAA, TXT, MX)
- *return*: list of records

modules/gethttp.py

def get_http(ip_address, hostname, int_port, usetls, http_pages)

Get_http has three main purposes: 1. check if the server automatically reroutes a port 80 connection (HTTP) to a port 443 connection (HTTPS), 2. follow redirects for the hostname, and 3. collects HTTP header information for the hostname. Connections are created using the http.client library, and redirects either generate a 200 status or terminate after 10 loops.

- *ip_address*: string IP address to connect to
- *hostname*: string hostname
- *int_port*: int port to connect to
- *usetls*: Boolean, true if function should connect using HTTPS (TLS), false for HTTP
- *http_pages*: Boolean if HTTP pages information should be included in output
- *return*: [host, path, http_to_https redirection], dictionary of HTTP data

def split_location(location)

Split_location receives a header location and splits it into protocol, domain name and path.

- *location*: header location

modules/gethttp2.py

def check_http2(domain_name, conn_port)

Check_http2 will return true if the selected application-layer protocol negotiation is h2 (HTTP/2), and false otherwise.

- *domain_name*: string hostname
- *conn_port*: port to connect to
- *return*: Boolean true if port supports HTTP2, false if not

Advanced Using the Cryptonice library in your own code.

Modules A description of each of the scanning and testing modules within Cryptonice.

CHAPTER 2

Indices and tables

- genindex