

KrustyLoader - Leveraging rust compilation artifacts to obtain reliable compilation timestamps and pivoting

 nofix.re/posts/2024-08-03-anti-rust/

KrustyLoader – Pivoting with deps. timestamp and compilation artifacts



Nofix

<https://nofix.re>

Mar 8, 2024

Compilation timestamps embedded in malware tend to be falsified by cyber-criminals. Rust compilation artifacts can give us another pivot could be a bit harder for malware developers to fake, although not impossible.

In my previous blog post, I talked about leveraging rust artifacts to get accurate symbol recovery on rust malwares, with the example of KrustyLoader.

One information we did not fully exploit is the crate's upload date.

`crates.io` exposes a lot of metadata about crates, such as versions, crate owners, associated GitHub repository, and upload time (for an example, see this).

`cargo` is a tool used in the Rust community to manage project dependencies. A cool thing about `cargo` is that it will try to pull the latest version available when adding a dependency to your project.

With that in mind, and given that your target has some dependencies, it isn't hard to use compilation artifacts to get a somewhat reliable idea of when your target malware development started.

For example, let's check some KrustyLoader sample

(816754f6eaf72d2e9c69fe09dcbe50576f7a052a1a450c2a19f01f57a6e13c17) dependencies:

```
$ rbi info 816754f6eaf72d2e9c69fe09dcbe50576f7a052a1a450c2a19f01f57a6e13c17
TargetRustInfo(
  rustc_version='1.70.0',
  rustc_commit_hash='90c541806f23a127002de5b4038be731ba1458ca',
  dependencies=[
    Crate(name='addr2line', version='0.17.0', features=[], repository=None),
    Crate(name='aes', version='0.7.5', features=[], repository=None),
    Crate(name='bytes', version='1.4.0', features=[], repository=None),
    Crate(name='cfb-mode', version='0.7.1', features=[], repository=None),
    Crate(name='futures-channel', version='0.3.28', features=[],
repository=None),
    Crate(name='futures-core', version='0.3.28', features=[], repository=None),
    Crate(name='futures-util', version='0.3.28', features=[], repository=None),
    Crate(name='generic-array', version='0.14.7', features=[], repository=None),
    Crate(name='getrandom', version='0.2.10', features=[], repository=None),
    Crate(name='gimli', version='0.26.2', features=[], repository=None),
    Crate(name='hashbrown', version='0.12.3', features=[], repository=None),
    Crate(name='hex', version='0.4.3', features=[], repository=None),
    Crate(name='http', version='0.2.9', features=[], repository=None),
    Crate(name='httparse', version='1.8.0', features=[], repository=None),
    Crate(name='hyper', version='0.14.27', features=[], repository=None),
    Crate(name='miniz_oxide', version='0.5.3', features=[], repository=None),
    Crate(name='num_cpus', version='1.15.0', features=[], repository=None),
    Crate(name='rand', version='0.8.5', features=[], repository=None),
    Crate(name='rand_chacha', version='0.3.1', features=[], repository=None),
    Crate(name='socket2', version='0.4.9', features=[], repository=None),
    Crate(name='tokio', version='1.29.0', features=[], repository=None),
    Crate(name='want', version='0.3.1', features=[], repository=None)
  ],
  rust_dependencies_import_hash='71d9b7211f83421909454a17a11c7d64',
  guessed_toolchain='linux-musl',
  guess_is_debug_build=False
)
```

Each of this dependency got added at some point in time. Since `bytes`, `http`, `httparse`, `http-body`, `futures*`, `tokio`, `socket2` and `want` are dependencies of `hyper`, I assume they all got added at the same time by cargo after the developer ran `cargo add hyper`.

At this point, `cargo` probably checked out the latest version of `hyper`, along with its dependencies. At the time of running the command, `hyper`'s version probably was `0.14.27`. According to crates.io, this version was published on `2023-04-13T20:10:42.655568+00:00`. The following version was published 3 months later.

Latest `tokio`'s version at this time probably was `1.29.0`, which was published on `2023-05-27T21:19:40.446763+00:00`. Its following version, `1.29.1` was published on `"2023-06-29T22:04:27.541298+00:00"`, merely a month later.

Since tokio's version used in this sample was added after **2023-05-27**, but before **2023-06-29**, we can be confident about the fact that the malware started to get developed between those two dates.

We can repeat the process for all dependencies, and find the one that was published more recently, as well as find the first dependency that had a new version that do not appear in our target.

Using `rbi`:

```
$ rbi guess_timestamp
816754f6eaf72d2e9c69fe09dcbe50576f7a052a1a450c2a19f01f57a6e13c17
Latest dependency was added between 2023-06-27 20:37:41.423254+00:00 and 2023-06-29
22:04:27.541298+00:00
```

With that we can confidently say that the developement of this sample started between 27 and 29 of June 2023.

Pivots

To find more similar samples, one could search for the project name or username that appears in the strings of the target.

You could also search for specific dependencies version. For example, searching on hybrid-analysis for the following set of strings:

- hyper-0.14.27
- want-0.3.1
- tokio-1.29.0
- socket2-0.4.9
- http-0.2.9
- futures-util-0.3.28

reveals many KrustyLoader samples, notably ELF-32bits, PE32+ (seemingly compiled with MinGW) and ARM samples, all having almost the same dependencies, revealing that they probably come from the same rust project.

Those are samples that Synacktiv's Yara rule miss due to the change of architecture and compiler.

Configuration extractor

Here is a modified version of Synacktiv's extractor that supports PE files:

```

#!/usr/bin/env python3
import pathlib
import sys
from binascii import unhexlify

from Crypto.Cipher import AES
from Crypto.Hash import SHA256

def xor(a,b):
    return bytes([x^b for x in a])

def extract_configuration(sample: pathlib.Path):
    data = sample.read_bytes()
    artifacts = [b'/tmp/', b'c:/windows/temp/']
    ENCRYPTED = None

    h = SHA256.new()
    h.update(data)

    print(f"Sample SHA256sum: {h.hexdigest()}")

    end = data.find(b"||||||||||||||||")
    start = end - 0x100

    for artifact in artifacts:
        start = start + data[start:end].find(artifact) + len(artifact)
        try:
            ENCRYPTED = unhexlify(data[start:end])

        except:
            continue

    assert ENCRYPTED is not None

    # 40 80 f5 XX == xor bpl, XX
    before_xorkey = data.find(bytes.fromhex("FFFF4080F5"))
    XORKEY = data[before_xorkey+len(bytes.fromhex("FFFF4080F5"))]
    print(f"XOR KEY: {hex(XORKEY)}")
    encrypted_stage2 = xor(ENCRYPTED, XORKEY)

    start = start - len(artifact) - 32
    AESKEY = data[start:start+16]

    start += 16
    AESIV = data[start:start+16]

    SEGMENT_SIZE = 128

    print(f"AES-128 CFB KEY: {AESKEY.hex()}")
    print(f"AES-128 CFB IV: {AESIV.hex()}")
    cipher = AES.new(AESKEY, AES.MODE_CFB, iv=AESIV, segment_size=SEGMENT_SIZE)

```

```

        decrypted = cipher.decrypt(encrypted_stage2)
        print(f"Decrypted Stage Hoster URL: {decrypted}")

if __name__ == "__main__":

    if len(sys.argv) < 2:
        print("usage: python crusty_decrypto.py ./sample")
        exit()

    extract_configuration(pathlib.Path(sys.argv[1]))

```

Interestingly, the ARM samples I found had no string obfuscation and had full symbols. The configuration was in clear text right in the sample. Thus, I did not adapt the configuration extractor for those.

IOCs

The following hashes come from [this search](#) on hybrid-analysis.

Sample	File type
9676c7a7a75ee08de77b7d1c24a1ae97f507af08fb5c5ac6d02973777e4e232f	PE32+ (no symbols)
4feb3dcfe57e3b112568ddd1897b68aeb134ef8addd27b660530442ea1e49cbb	ELF 64 (aarch64, symbols)
49062378ab3e4a0d78c6db662efb4dbc680808fb75834b4674809bc8903adaea	ELF 32 (symbols)