# How to Write a Hancitor Extractor in Go

pid4.io/posts/how_to_write_a_hancitor_extractor/

3 minutes

Hancitor (AKA Chanitor) is a well known and old malware family. A number of great blogs have been posted on the topic of analyzing the loader. Recently @herrcor live streamed building an extractor with Python. It was a great session and worth checking out.

It seems that a Python extractor is relatively straightforward. So you might ask: Can we build the same extractor with Go, *using only the standard libary*? Well, Yes, but with some drawbacks.

Here's the Gist if you want to skip the comments and check out the code.

Let's see what can be done *without importing any external code*. Below we see that `Extract` is the only exported function, and it returns a configuration `struct`.

```
// Config represents a parsed malware configuration
type Config struct {
        Url        []url.URL
        CampaignID string
        Family     string
        Raw        json.RawMessage // any data that is not represented elsewhere in
the struct can be put here
}

// Extract extracts configuration a HANCITOR PE
func Extract(mal *pe.File) (config.Config, error) {
        if mal.Section(".data") == nil {
                return config.Config{}, errors.New("invalid pointer to PE data
section")
        }
        dataSection, err := mal.Section(".data").Data()
        if err != nil {
                return config.Config{}, err
        }
```

The `"debug/pe"` library was certainly not intended for malware analysis, but it is sufficient for this task. The task being to parse out of the data the configuration material.

The disadvantage to using only the standard library is that `"debug/pe"` does not offer the ability to do dynamic offset calculations. In lieu of using dynamic offsets as shown by herrcor, CAPEv2 can be used as a template.

```
// key material at 16:24 in the data section
hash := sha1.Sum(dataSection[16:24])
rc4Key := hash[:5]

// 24 == starting after the key material
// 2000 == total size of config
conf, err := decryptConfig(rc4Key, dataSection[24:2000])
if err != nil {
        return config.Config{}, err
}
```

After the key material is parsed we can decrypt and parse the configuration values.

```
cipher, err := rc4.NewCipher(rc4Key)
if err != nil {
        return config.Config{}, err
}

// decrypt the config
cipher.XORKeyStream(ciphertext, ciphertext)

// start parsing into a Config struct
var conf config.Config
build := buildID(ciphertext)
r, err := json.Marshal(fmt.Sprintf(`{"rc4_key": "%v" }`, rc4Key))
if err != nil {
        return config.Config{}, err
}
```

So it is possible to write a full extractor in Go instead of Python. But there are some clear drawbacks. On the other hand we didn't take a trip through dependency hell to get to a statically compiled CLI tool, which didn't import a single third party library. So that was nice.

The full source code for the extractor can be found as Gist . Just call the exported function like so: `config, err := hancitor.Extract(mal)` in which `mal` is a parsed DLL parsed as a *pe.File .

---

malware analysis go programming

454 Words

2021-10-05

---