XXEs in Golang are Surprisingly Hard

Eric Payne

August 2020

Introduction

- AppSec is all about digging into strange behaviour in an application

- Digging into XXEs in Golang can bring us some surprising results!

Eric Payne

Software Engineer

Clio - Cloud-Based Legal Technology • University of Toronto

Experience

Application Security Developer
Clio - Cloud-Based Legal Technology
Sep 2019 - Present • 1 yr

Volunteering

Volunteer Organizer
OWASP Foundation
2019 - Present

What is an XML External Entity Attack?

- XXEs rank **number four** in the OWASP Top Ten security risks
- XXEs exploit XML parsers with overly permissive configurations
- XXEs can be used to cause:
  - local file inclusion
  - server-side request forgery
  - port scanning
  - and more!

Sample Vulnerable Ruby Application

```ruby
require 'sinatra'
require 'nokogiri'


post '/parse-xml' do
    xml_payload = params[:xml_payload]
    parsed_xml = Nokogiri::XML(xml_payload) do |config|
        config.options = Nokogiri::XML::ParseOptions::NOENT
    end
    return "#{parsed_xml}"
end
```

Simple XML Payload (no exploit)

```
<?xml version="1.0"?>
<root>
   <child>some benign content</child>
</root>
```

```
<?xml version="1.0"?>
<root>
   <child>some benign content</child>
</root>
```

Payload with Entity Definition (no exploit)

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY example_entity "foo">]>
<root>
   <child>&example_entity;</child>
</root>
```

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY example_entity "foo">]>
<root>
   <child>foo</child>
</root>
```

Payload with External Entity Defined (exploit!)

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY example_entity SYSTEM 'file:///etc/passwd'>]>
<root>
   <child>&example_entity;</child>
</root>
```

Payload with External Entity Defined (exploit!)

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY example_entity SYSTEM "file:///etc/passwd">]>
<root>
   <child>##
# User Database
#
# Note that this file is consulted directly only when the system is running
# in single-user mode.  At other times this information is provided by
 ...
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
...
```

Can we achieve an XXE in a Go application?

- Just focus on our XML parsing for now; don't need a REST endpoint in this example

- Go implements a lot of useful functionality in its default libraries; let's use Go's `"encoding/xml"` library for our XML parsing

First Attempt

- We decode our XML content into a strongly typed `Root struct`
- With `decoder.Strict = false`:

```
go run firstattempt.go
XML parsed into struct: {&example;}
```

- With `decoder.Strict = true`:

```
go run firstattempt.go

Error: XML syntax error on line 5:
invalid character entity &example;

exit status 1
```

```
import (
    "encoding/xml"
)

type Root struct {
    Child string `xml:"child"`
}

func main() {
    document := `
    <?xml version="1.0"?>
    <!DOCTYPE root [<!ENTITY example SYSTEM
'file:///etc/passwd'>]>
    <root>
        <child>&example;</child>
    </root>`
    decoded := Root{}
    decoder := xml.NewDecoder(strings.NewReader(document))
    decoder.Strict = false

    if err := decoder.Decode(&decoded); err != nil {
        fmt.Println("Error:", err)
        os.Exit(1)
    }
    fmt.Printf("XML parsed into struct: %v \n", decoded)
}
```

Use the “entity map” interface

```
// Entity can be used to map non-standard entity names to string replacements.
// The parser behaves as if these standard mappings are present in the map,
// regardless of the actual map content:
//
//      "lt": "<",
//      "gt": ">",
//      "amp": "&",
//      "apos": "'",
//      "quot": `"`,
Entity map[string]string
```

```
decoder.Entity = make(map[string]string, 128)
decoder.Entity["example"] = "SYSTEM 'file:///etc/passwd'"
```

```
go run entitymap.go
XML parsed into struct: {SYSTEM 'file:///etc/passwd'}
```

golang / go

Watch 3.5k | Star 75.6k | Fork 10.9k

Code | Issues 5k+ | Pull requests 222 | Actions | Projects 1 | Wiki | Security | Insights

master | go / src / encoding / xml /

Go to file | Add file

ianlancetaylor Revert "encoding/xml: fix reserved namespace check to be case-insensi... ... 2ca0f5a on Jun 29 History

..		
atom_test.go	encoding/xml: improve package based on the suggestions from metalinter	3 years ago
example_marshaling_test.go	encoding/xml, encoding/json: docs and examples using custom marshalers	2 years ago
example_test.go	all: make copyright headers consistent with one space after period	5 years ago
example_text_marshaling_test.go	encoding/xml, encoding/json: docs and examples using custom marshalers	2 years ago
marshal.go	Revert "encoding/xml: fix reserved namespace check to be case-insensi...	2 months ago
marshal_test.go	Revert "encoding/xml: fix reserved namespace check to be case-insensi...	2 months ago
read.go	encoding/xml: only initialize nil struct fields when decoding	3 months ago
read_test.go	all: change github.com issue links to golang.org	3 years ago
typeinfo.go	encoding/xml: only initialize nil struct fields when decoding	3 months ago
xml.go	all: avoid string(i) where i has type int	6 months ago
xml_test.go	encoding/xml: fix token decoder on early EOF	10 months ago

Library code review

- This short code block in `xml.go` is the only reference we see to `.Entity`
- We don’t see any calls to `os.Open()` or `http.Get()`
- All that `"encoding/xml"` does with its entity map is a simple string substitution!

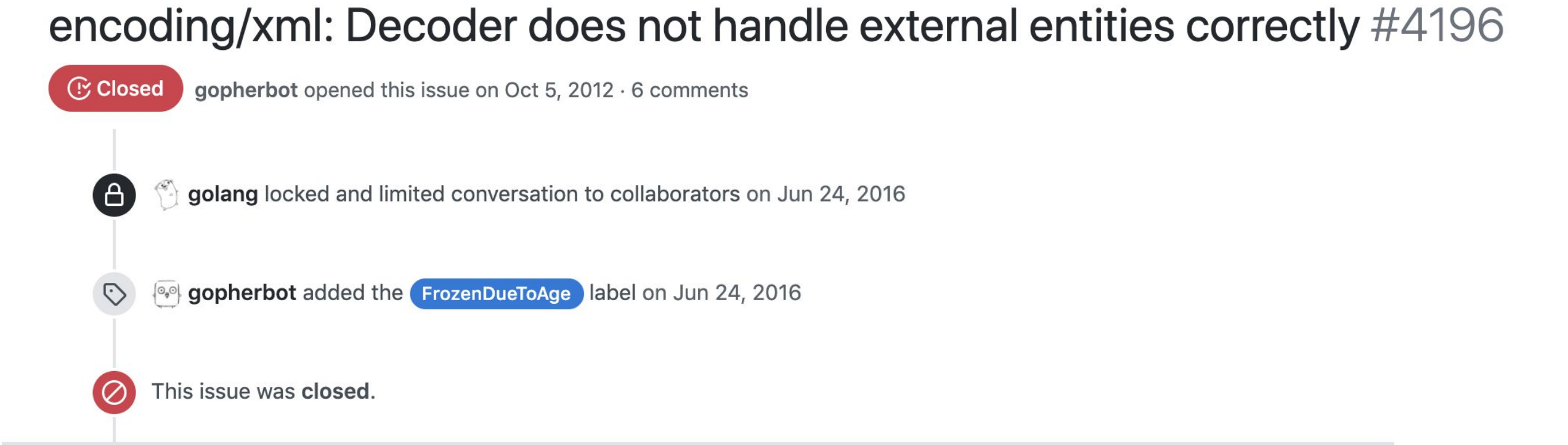
```
if r, ok := entity[s]; ok {
    text = string(r)
    haveText = true
 } else if d.Entity != nil {
    text, haveText = d.Entity[s]
}
```

“Surprisingly hard”, not impossible

- The default choice for simple XML parsing *doesn’t actually implement external entities*

- **“Surprisingly hard”, not impossible:** there are Go bindings available for robust XML parsers like `libxml2`, so Go applications *may* still be vulnerable to this

Not clearly documented behaviour



Conclusion

- Not clearly documented in Go reference material; unclear if this decision was made from a *security* or *dev effort* standpoint

- Secure defaults in the Golang ecosystem: by default, a Go application doesn’t need to worry about XML External Entity attacks


OWASP