



시큐어 코딩 가이드


스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드

Institute: 고려대학교 블록체인 연구센터

Date: November. 15, 2022


Version: 1.0



 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

Contents

1장. 개요	3
2장. 시큐어코딩 가이드	4
1. 입력데이터 검증 및 표현	4
1. 잘못된 입력으로 제공된 HTTP 요청	4
2. 형식 문자열/문자열 연결을 사용한 SQL 쿼리 구성	6
3. /debug/pprof에 엔드포인트가 자동으로 노출되는 프로파일링	8
4. HTML 템플릿에서 탈출하지 않은 데이터 사용	9
5. 사용자가 제어하는 소스에서 명령 주입	11
6. 잘못된 파일 권한 사용	13
7. 잘못된 파일 경로 입력	15
8. 압축 해제 시 파일 경로 설정	17
2. 보안기능	19
1. 하드 코딩된 자격 증명 찾기	19
2. 모든 인터페이스에 바인딩	20
3. unsafe 패키지에 대한 함수 호출	22
4. ssh.InsecureIgnoreHostKey 함수 사용	24
5. 오버플로우	27
6. DoS (Denial of Service)	29
7. 예측 가능한 경로를 사용한 임시 파일 생성	31
8. 잘못된 TLS 설정	33
9. 암호 알고리즘 키 길이 부족	35
10. 난수 생성	37
3. 에러처리	39
1. 확인되지 않은 오류	39
4. Import 블랙리스트	41
1. crypto/md5	41
2. crypto/des	42
3. crypto/rc4	44
4. net/http/cgi	45
5. crypto/sha1	46

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide


1장. 개요

디지털 경제가 일상생활화 되면서 블록체인과 스마트 컨트랙트에 대한 관심과 활용이 계속 늘고 있어, 머지 않은 미래에서는 블록체인과 이를 활용한 스마트 컨트랙트가 디지털 경제의 한 핵심요소가 될 것으로 예상 된다. 스마트 컨트랙트는 정상 작동 중에 오류로 인해 재정상의 피해를 볼 수도 있고, 공격자의 공격으로 인해 재정상의 피해를 볼 수 있다. 이에 따라 고신뢰성, 고안전성, 고보안성이 중요성이 대두됨에 따라 현재 고려대학교 블록체인 연구소에서 제시한 Solidity 시큐어코딩 가이드와 같은 것을 참고하여 개발 시 보안 위협을 최소화하여 소프트웨어를 개발하는 것이 중요하다.

본 가이드는 개발 시 보안 위협을 최소화 하기 위해 Go 언어로 소프트웨어 개발시 예방 해야하는 보안 약점들에 대한 설명을 제시하고, 좋지 않은 예, 좋은 예를 제시하여 안전한 소프트웨어 개발에 도움이 되도록 이바지한다.

본 가이드의 목적, 구성 및 활용은 아래와 같다.

목적	안전한 Go 소프트웨어 개발을 위한 시큐어 코딩 가이드 제시
구성	1장. 개요 2장. 시큐어코딩 가이드 2-1. 입력데이터 검증 및 표현 2-2. 보안기능 2-3. 에러처리 2-4. Import 블랙리스트
활용	Go 개발 시 시큐어 코딩을 적용하여 안전한 소프트웨어 개발 Go 개발 시 고려해야하는 보안 약점 및 안전한 개발 방법론에 대한 이해

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

2장. 시큐어코딩 가이드

1. 입력데이터 검증 및 표현

사용자 입력값, 프로그램 입력 값에 대한 충분한 검증을 하지 않고, 지정된 표현이 아닌 잘 못된 표현 등으로 인한 보안약점이 발생하여, 의도치 않은 문제점이 야기 될 수 있다.

1. 잘못된 입력으로 제공된 HTTP 요청

사용자 입력과 같은 신뢰할 수 없는 소스에서 URL을 가져오면 공격자가 응용 프로그램을 잘못된 웹 사이트로 리다이렉션하고 추가 공격을 수행할 수 있다. Get() 함수는 지정된 URL에 GET을 발행하며, 결과가 적절하면 클라이언트의 CheckRedirect 함수를 호출한 후 GET이 리다이렉션에 따른다. 즉, 공격자가 응용 프로그램을 다양한 곳으로 보낼 수 있다.


예시

- Bad

```
package main

import (
    "net/http"
    "io/ioutil"
    "fmt"
    "os"
)

func main() {
    url := os.Getenv("tainted_url")
    resp, err := http.Get(url)
    if err != nil {
        panic(err)
    }
    defer resp.Body.Close()
    body, err := ioutil.ReadAll(resp.Body)
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```

    if err != nil {
        panic(err)
    }
    fmt.Printf("%s", body)
}

```

- Good

```

package main

import (
    "fmt"
    "net/http"
)


const url = "http://127.0.0.1"

func main() {
    resp, err := http.Get(url)
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(resp.Status)
}

```

참고문헌

- OWASP Top 10 Project, (2021), “OWASP Top Ten”,
<https://owasp.org/www-project-top-ten/>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

2. 형식 문자열/문자열 연결을 사용한 SQL 쿼리 구성

SQL 주입은 가장 많이 발생하는 보안 문제 중 하나이며 이로 인한 결과는 심각하다. fmt Golang 패키지의 형식 문자열 함수를 사용하여 SQL 쿼리를 동적으로 생성하면 SQL 주입 가능성을 쉽게 만들 수 있다. 왜냐하면 형식 문자열 함수가 특수 문자를 탈출하지 않고 형식 문자열에 두 번째 SQL 쿼리를 추가하기 때문이다.

예시

- Bad


```
package main

import (
    "database/sql"
    "fmt"
    "os"
)

func main(){
    db, err := sql.Open("sqlite3", ":memory:")
    if err != nil {
        panic(err)
    }
    q := fmt.Sprintf("SELECT * FROM foo where name = '%s'", os.Args[1])
    rows, err := db.Query(q)
    if err != nil {
        panic(err)
    }
    defer rows.Close()
}
```

- Good


```
package main
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

```
import (
    "database/sql"
)
const staticQuery = "SELECT * FROM foo WHERE age < 32"
func main(){
    db, err := sql.Open("sqlite3", ":memory:")
    if err != nil {
        panic(err)
    }
    rows, err := db.Query(staticQuery)
    if err != nil {
        panic(err)
    }
    defer rows.Close()
}
```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-564: SQL Injection: SQL Injection”, <https://cwe.mitre.org/data/definitions/564.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

3. /debug/pprof에 엔드포인트가 자동으로 노출되는 프로파일링

net/http/pprof를 가져오면 힙 프로파일 및 CPU 프로파일과 같은 중요한 어플리케이션 정보를 노출하는 디버그 엔드포인트가 자동으로 노출되어 어플리케이션 성능에도 영향을 미친다. 빌드에서 net/http/pprof import를 제거하는 것이 좋다.

예시


- Bad

```
package main

import (
    _ "net/http/pprof"
)
```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-200: Exposure of Sensitive Information to an Unauthorized Actor”, <https://cwe.mitre.org/data/definitions/200.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

4. HTML 템플릿에서 탈출하지 않은 데이터 사용

HTML 템플릿에 탈출하지 않은 데이터가 있을 수 있다. HTML을 자동으로 탈출하지 않으며 코드 주입 공격을 유발할 수 있으므로 탈출하지 않고 템플릿에서 외부 값을 사용하면 안된다.

- template.JS : JS를 사용하여 유효하지만 신뢰할 수 없는 JSON을 포함하는 것은 안전하지 않다. 대안은 json과 함께 JSON을 구문 분석하는 것이다. 자바스크립트 컨텍스트에서 제시될 때 sanitized JSON으로 변환될 템플릿으로 결과 개체를 unmarshal한 다음 전달한다.
- template.HTML, template.HTMLAttr, template.URL : 이 유형을 사용하면 보안 위험이 발생한다. 캡슐화된 내용은 템플릿 출력에 자세히 포함되므로 신뢰할 수 있는 소스에서 가져와야 한다.

예시


- Bad

```
package main

import (
    "fmt"
    "html/template"
    "os"
)

func main() {
    a := `{"name": "untrusted"}`
    t := template.Must(template.New("x").Parse(""))
    v := map[string]interface{}{
        "Body": template.JS(a),
    }

    if err := t.Execute(os.Stdout, v); err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```

    }
}

```

- Good

```

package main

import (
    "fmt"
    "html/template"
    "os"
)


func main() {
    t := template.Must(template.New("x").Parse(""))
    v := map[string]interface{}{
        "Body": template.JS(`{"name": "trusted"}`),
    }

    if err := t.Execute(os.Stdout, v); err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
}

```

참고문헌

- Common Weakness Enumeration, (2022), CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'),
<https://cwe.mitre.org/data/definitions/79.html>
- OWASP Top 10 Project, (2021), "OWASP Top Ten A03:2021 - Injection",
https://owasp.org/Top10/A03_2021-Injection/

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

5. 사용자가 제어하는 소스에서 명령 주입

충분한 검사 없이 사용자가 제공한 데이터로 작성된 명령 호출(os/exec)은 명령을 실행하여 데이터를 빼내거나 시스템을 손상시킬 수 있다. 사용자가 제공한 데이터를 명령 호출에서 직접 사용하지 않도록 하거나 사용하기 전에 삭제하는 것이 좋다.

- 명령어를 작성하기 전에 사용자 입력 검사
- 하드 코딩된 문자열 리터럴을 사용하여 실행할 명령 지정
- 사용자 입력에 따라 신뢰할 수 있는 하드 코딩된 명령 선택

예시

- Bad


```
func executor(req *http.Request) {
    cmdQ := req.URL.Query()["cmd"][0]

    cmd := exec.Command(cmdQ)
    cmd.Run()
}
```

- Good

```
func executor(req *http.Request) {
    cmdQ := req.URL.Query()["cmd"][0]

    switch cmdQ {
    case "a":
    case "b":
        cmdQ = "app2"
    case "<something unsafe":
        cmdQ = "true"
    default:
    }
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```

cmd := exec.Command(cmdQ)


cmd.Run()

}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')”, <https://cwe.mitre.org/data/definitions/78.html>
- OWASP Top 10 Project, (2021), “OWASP Top Ten A03:2021 - Injection”, https://owasp.org/Top10/A03_2021-Injection/

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

6. 잘못된 파일 권한 사용

일반적으로 모든 보안 규칙은 최소 권한의 원칙을 따른다. 단, 디렉토리를 작성하는 사용자가 아닌 다른 사용자가 디렉터리에 접근해야하는 경우는 제외한다.

예시

- Bad

```
package main

import (
    "fmt"
    "os"
)


func main() {
    err := os.Mkdir("/tmp/mydir", 0777)
    if err != nil {
        fmt.Println("Error when creating a directory!")
        return
    }
}
```

- Good

```
package main

import (
    "fmt"
    "os"
)


func main() {
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```
err := os.Mkdir("/tmp/mydir", 0600)
if err != nil {
    fmt.Println("Error when creating a directory!")
    return
}
}
```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')”, <https://cwe.mitre.org/data/definitions/78.html>
- Common Weakness Enumeration, (2022), “CWE-276: Incorrect Default Permissions”, <https://cwe.mitre.org/data/definitions/276.html>
- OWASP Top 10 Project, (2021), “OWASP Top Ten A05:2021 - Security Misconfiguration”, https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

7. 잘못된 파일 경로 입력

파일 경로를 변수 입력으로 받아 공격자가 수정하여 제어할 수 있다.

예시

- Bad

```
package main


import (
    "fmt"
    "io/ioutil"
    "strings"
)

func main() {
    repoFile := "/safe/path/../../private/path"
    if !strings.HasPrefix(repoFile, "/safe/path/") {
        panic(fmt.Errorf("Unsafe input"))
    }
    byContext, err := ioutil.ReadFile(repoFile)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s", string(byContext))
}
```

- Good

```
package main

import (
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```


"fmt"
"io/ioutil"
"path/filepath"
"strings"
)

func main() {
    repoFile := "/safe/path/../../private/path"
    repoFile = filepath.Clean(repoFile)
    if !strings.HasPrefix(repoFile, "/safe/path/") {
        panic(fmt.Errorf("Unsafe input"))
    }
    byContext, err := ioutil.ReadFile(repoFile)
    if err != nil {
        panic(err)
    }
    fmt.Printf("%s", string(byContext))
}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')”, <https://cwe.mitre.org/data/definitions/22.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

8. 압축 해제시 파일 경로 설정

파일을 참조하는 변수를 조작하거나 절대 파일 경로를 사용하면 응용 프로그램 소스 코드 또는 구성 및 중요한 시스템 파일을 포함하여 파일 시스템에 저장된 임의의 파일 및 디렉토리에 액세스가능 하므로 “..”을 사용하지 않아야 한다.

예시

- Bad

```
package main


import (
    "archive/zip"
    "io/ioutil"
    "path/filepath"
)

func unzip(f string) {
    r, _ := zip.OpenReader(f)
    for _, f := range r.File {
        p, _ := filepath.Abs(f.Name)
        ioutil.WriteFile(p, []byte("dummy"), 0666)
    }
}
```

- Good

```
package main

import (
    "archive/zip"
    "io/ioutil"
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```


"path/filepath"
"strings"
)

func unzipGood(f string) {
    r, _ := zip.OpenReader(f)
    for _, f := range r.File {
        p, _ := filepath.Abs(f.Name)
        if !strings.Contains(p, "..") {
            ioutil.WriteFile(p, []byte("present"), 0666)
        }
    }
}

```

참고문헌

- OWASP Top 10 Project, (2021), “OWASP Top Ten A03:2021 - Injection”, https://owasp.org/Top10/A03_2021-Injection/
- Common Weakness Enumeration, (2022), “CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')”, <https://cwe.mitre.org/data/definitions/22.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

2. 보안기능

인증, 접근제어, 암호화 등 보안기능을 부적절하게 구현하여 의도치 않은 문제점이 야기 될 수 있다.

1. 하드 코딩된 자격 증명 찾기

하드 코딩된 암호를 사용하면 암호 추측 가능성이 매우 높아지기 때문에 악의적인 사용자가 해당 계정을 통해 접근할 수 있다. 따라서 하드코딩을 하면 안된다.

예시

- Bad

```
package main


import "fmt"

func main() {
    username := "admin"
    var password = "f62e5bcda4fae4f82370da0c6f20697b8f8447ef"

    fmt.Println("Doing something with: ", username, password)
}
```

참고문헌

- Common Weakness Enumeration, (2022), "CWE-259: Use of Hard-coded Password", <https://cwe.mitre.org/data/definitions/259.html>
- Common Weakness Enumeration, (2022), "CWE-321: Use of Hard-coded Cryptographic Key", <https://cwe.mitre.org/data/definitions/321.html>
- Common Weakness Enumeration, (2022), "CWE-798: Use of Hard-coded Credentials", <https://cwe.mitre.org/data/definitions/798.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

2. 모든 인터페이스에 바인딩

모든 네트워크 인터페이스에 바인딩하면 의도하지 않은 인터페이스 트래픽에 대한 서비스를 열 수 있다.

IP 주소로 "0.0.0.0"을 사용하여 포트를 모든 인터페이스에 바인딩할 때 기본적으로 라우팅을 통해 소켓에 도달할 수 있는 경우 IPv4 주소의 연결을 허용해야 한다. 따라서 모든 인터페이스에 바인딩하는 것은 보안 위험과 관련이 있으므로 권장되지 않는다.

예시

- Bad

```
package main


import (
    "log"
    "net"
)

func main() {
    l, err := net.Listen("tcp", "0.0.0.0:2000")
    if err != nil {
        log.Fatal(err)
    }
    defer l.Close()
}
```

- Good

```
package main

import (
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

```


    "log"
    "net"
  )

func main() {
    l, err := net.Listen("tcp", "1.2.3.4:2000")
    if err != nil {
        log.Fatal(err)
    }
    defer l.Close()
}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-200: Exposure of Sensitive Information to an Unauthorized Actor”, <https://cwe.mitre.org/data/definitions/200.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

3. unsafe 패키지에 대한 함수 호출

Go에서 unsafe 패키지를 사용하면 낮은 수준의 메모리 관리 및 C 언어의 많은 강점을 제공할 수 있지만 공격자에게 유연성을 제공한다.

예시

- Bad


```
package main

import (
    "fmt"
    "unsafe"
)

type Fake struct{}

func (Fake) Good() {}

func main() {
    unsafeM := Fake{}
    unsafeM.Good()
    intArray := [...]int{1, 2}
    fmt.Printf("
intArray: %v
", intArray)
    intPtr := &intArray[0]
    fmt.Printf("
intPtr=%p, *intPtr=%d.
", intPtr, *intPtr)
    addressHolder := uintptr(unsafe.Pointer(intPtr)) + unsafe.Sizeof(intArray[0])
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

```


    intPtr = (*int)(unsafe.Pointer(addressHolder))
    fmt.Printf("
intPtr=%p, *intPtr=%d.

", intPtr, *intPtr)
}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-242: Use of Inherently Dangerous Function” <https://cwe.mitre.org/data/definitions/242.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

4. ssh.InsecureIgnoreHostKey 함수 사용

'InsecureIgnoreHostKey'는 모든 호스트 키를 허용하는데 사용된다. 그래서 생산 코드에 사용해서는 안 된다.

ssh.secureIgnoreHostKey()에서 함수는 호스트 키 검사 처리 기능이 없으므로 안전하지 않은 nil 값을 반환한다. 생산에서 ssh.InsecureIgnoreHostKey() 함수를 사용하도록 클라이언트를 구성하지 않는 것이 좋다. 대신 보안 설정을 통해 서버 스푸핑 또는 중간자 공격을 방지해야 한다.

예시

- Bad

```
package main

import (
    "golang.org/x/crypto/ssh"
)

func main() {
    isc := &ssh.ClientConfig{
        User:      "user",
        Auth:      []ssh.AuthMethod{ssh.Password("pass")},
        HostKeyCallback: ssh.InsecureIgnoreHostKey(),
    }
}
```

- Good

```
package main

import (
    "encoding/base64"
    "fmt"
)
```



```

"net"

"os"

"golang.org/x/crypto/ssh"
)


func keyString(pk ssh.PublicKey) string {
    return pk.Type() + " " + base64.StdEncoding.EncodeToString(pk.Marshal())
}

func TrustedHostKeyCallback(key string) ssh.HostKeyCallback {
    if key == "" {
        return func(_ string, _ net.Addr, k ssh.PublicKey) error {
            fmt.Fprintf(os.Stderr, "[WARN] SSH key verification is not in effect
(Fix by adding trustedKey: %q)
", keyString(k))
            return nil
        }
    }

    return func(_ string, _ net.Addr, k ssh.PublicKey) error {
        if ks := keyString(k); key != ks {
            return fmt.Errorf("[ERROR] SSH key verification: expected %q but got
%q", key, ks)
        }
        return nil
    }
}

func main() {
    // Server Key
    hostKey := "dummy-host-key"

```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	


```

// SSH's Client Configuration
ssc := &ssh.ClientConfig{
    User:          "user",
    Auth:          []ssh.AuthMethod{ssh.Password("pass")},
    HostKeyCallback: TrustedHostKeyCallback(hostKey),
}
}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-322: Key Exchange without Entity Authentication”, <https://cwe.mitre.org/data/definitions/322.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

5. 오버플로우

큰 개체에 대해 연산을 수행하면 오버플로우 되거나 랩어라운드 될 수 있다. 오버플로우로 인해 연산 결과가 음수가 되고 부호 없는 타입의 결과를 더 작은 수가 된다. 연산된 크기로 할당하면 크기가 음수이거나 버퍼 크기가 예기치 않게 작아서 런타임 오류가 발생할 수 있다. 따라서 이러한 연산으로부터 보호하는 것이 좋다.

예시

- Bad

```
func encoder(v interface{}) ([]byte, error) {
    data, err := json.Marshal(v)
    if err != nil {
        return nil, err
    }


    size := len(data) + (len(data) % 16)

    buffer := make([]byte, 0, size)
    copy(buffer, data)

    return data, nil
}
```

- Good

```
func encoder(v interface{}) ([]byte, error) {
    data, err := json.Marshal(v)
    if err != nil {
        return nil, err
    }
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```

if len(data) > 64*1024*1024 {
    return nil, errors.New("too large")
}

size := len(data) + (len(data) % 16)


buffer := make([]byte, 0, size)
copy(buffer, data)

return data, nil
}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-121: Stack-based Buffer Overflow”, <https://cwe.mitre.org/data/definitions/121.html>
- Common Weakness Enumeration, (2022), “CWE-190: Integer Overflow or Wraparound”, <https://cwe.mitre.org/data/definitions/190.html>
- Common Weakness Enumeration, (2022), “CWE-680: Integer Overflow to Buffer Overflow”, <https://cwe.mitre.org/data/definitions/680.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

6. DoS (Denial of Service)

악의적으로 조작된 아카이브는 압축 해제에 대한 대량의 데이터를 생성하여 응용 프로그램에 대한 서비스 거부(DoS) 공격을 발생시킬 수 있다. 단적인 예로, 압축 해제를 할 때 복사된 바이트 수를 제어하려면 `io.Copy`와 `io.CopyBuffer` 대신 `io.CopyN`을 사용하는 것이 좋다.

예시

- Bad


```
package main

import (
    "bytes"
    "compress/zlib"
    "io"
    "os"
)

func foo(b io.Reader) {
    r, err := zlib.NewReader(b)
    if err != nil {
        panic(err)
    }

    _, err = io.Copy(os.Stdout, r)
    if err != nil {
        panic(err)
    }

    r.Close()
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

- Good

```
package main

import (
    "bytes"
    "compress/zlib"
    "io"
    "os"
)


func foo(b io.Reader) {
    r, err := zlib.NewReader(b)
    if err != nil {
        panic(err)
    }

    _, err = io.CopyN(os.Stdout, r, 1024) // "n" may change depending on the
    application
    if err != nil {
        panic(err)
    }

    r.Close()
}
```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-409: Improper Handling of Highly Compressed Data (Data Amplification)”, <https://cwe.mitre.org/data/definitions/409.html>
- OWASP Foundation, (2022), “Denial of Service”, https://owasp.org/www-community/attacks/Denial_of_Service

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

7. 예측 가능한 경로를 사용한 임시 파일 생성

os.TempDir, ioutil.TempDir 및 ioutil.TempFile()을 사용하여 생성된 파일 경로가 올바르고 예측 불가능한지 확인한다. 임시 파일이 포함된 디렉터리에 파일 이름을 예측하고 쓸 수 있는 악의적인 사용자는 프로그램이 파일 자체를 만들기 전에 임시 파일의 이름으로 심볼 링크를 만들어 임시 파일을 효과적으로 가로챌 수 있다. 이를 통해 악의적인 사용자가 악의적인 데이터를 제공하거나 프로그램이 선택한 파일에 영향을 미칠 수 있다. 임시 파일을 만들 때 권장되는 방법으로 os.TempDir()/ioutil.TempDir()을 사용하여 디렉터리와 ioutil의 이름을 가져오거나 os.Create를 사용해야 한다.

예시

- Bad

```
package samples

import (
    "fmt"
    "io/ioutil"
)


func main() {
    err := ioutil.WriteFile("/tmp/demo2", []byte("This is some data"), 0644)
    if err != nil {
        fmt.Println("Error while writing!")
    }
}
```

- Good

```
package main

import (
    "fmt"

```

<div></div> <div>블록체인연구소</div> <div>Blockchain Research Institute</div>	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

```

"io/ioutil"
"os"
)

func main() {
    content := []byte("This is some data")

    dir, err := ioutil.TempDir(os.TempDir(), "")
    if err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }


    tmp, err := ioutil.TempFile(dir, "demo2")
    if err != nil {
        fmt.Fprintln(os.Stderr, err)
        os.Exit(1)
    }
    defer tmp.Close()

    _, _ = tmp.Write(content)
}

```

참고문헌

- Common Weakness Enumeration, (2022), “CWE-377: Insecure Temporary File”, <https://cwe.mitre.org/data/definitions/377.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
소속	고려대학교 / 블록체인 연구소	
제목	Go Secure Coding Guide	

8. 잘못된 TLS 설정

낮은 버전의 TLS를 설정하거나, TLS 버전에 대한 충분한 명시가 이뤄지지 않을 때, 공격자로부터 공격을 받을 수 있어, 최신 버전의 TLS를 사용하고, TLS의 버전을 명시해야 한다.

예시

- Bad

```
package main

import "crypto/tls"

func main() {
    config := &tls.Config{MinVersion: 0}
    ...
}
```


- Good

```
package main


import "crypto/tls"

func saferTLSConfig() {
    config := &tls.Config{}
    config.MinVersion = tls.VersionTLS12
    config.MaxVersion = tls.VersionTLS13
    // (or)
    config.MaxVersion = 0
}
```

참고문헌

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

- OWASP Top 10 Project, (2021), “OWASP Top Ten A02:2021 - Cryptographic Failures”, https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- OWASP Top 10 Project, (2021), “OWASP Top Ten A05:2021 - Security Misconfiguration”, https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- Common Weakness Enumeration, (2022), “CWE-295: Improper Certificate Validation”, <https://cwe.mitre.org/data/definitions/295.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

9. 암호 알고리즘 키 길이 부족

키 길이가 짧으면 전사 공격으로 인하여 알고리즘이 깨질 수 있어, 충분한 키 길이를 가진 암호 알고리즘을 사용하여야 한다.


예시

- Bad

```
package main
import (
    "crypto/rand"
    "crypto/rsa"
    "fmt"
)
func main() {
    pvk, err := rsa.GenerateKey(rand.Reader, 1024)
    if err != nil {
        fmt.Println(err)
    }
    fmt.Println(pvk)
}
```

- Good

```
package main
import (
    "crypto/rand"
    "crypto/rsa"
    "fmt"
)
func main() {
    pvk, err := rsa.GenerateKey(rand.Reader, 2048)
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide


```

if err != nil {
    fmt.Println(err)
}
fmt.Println(pvk)
}

```

참고문헌

- OWASP Top 10 Project, (2021), “OWASP Top Ten A02:2021 - Cryptographic Failures”, https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- OWASP Top 10 Project, (2021), “OWASP Top Ten A05:2021 - Security Misconfiguration”, https://owasp.org/Top10/A05_2021-Security_Misconfiguration/
- Common Weakness Enumeration, (2022), “CWE-310: Cryptographic Issues”, <https://cwe.mitre.org/data/definitions/310.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

10. 난수 생성

math/rand 난수 생성에 보안성을 제공해주지 않으므로, 난수 생성에 보안성을 제공하는 crypto/rand를 사용해야 한다.

예시

- Bad

```
package main

import "math/rand"

func main() {
    bad := rand.Int()
    println(bad)
}
```

- Good


```
package main

import "crypto/rand"

func main() {
    good, _ := rand.Read(nil)
    println(good)
}
```


참고문헌

- OWASP Top 10 Project, (2021), "OWASP Top Ten A02:2021 - Cryptographic Failures", https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- OWASP Top 10 Project, (2021), "OWASP Top Ten A05:2021 - Security Misconfiguration",

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

- Common Weakness Enumeration, (2022), “CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)”,
<https://cwe.mitre.org/data/definitions/338.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

3. 에러처리

부적절한 에러 처리 및 예외 처리를 하여 의도치 않은 문제점을 야기할 수 있다.

1. 확인되지 않은 오류

Go에서 결과 튜플과 함수에서 오류 값을 반환하는 기능은 유용하지만 오류 값을 확인하기 전까지 함수의 결과가 안전하지 않다. 오류 값을 확인하지 않으면 많은 공격을 받을 수 있다.

예시


- Bad

```
scanner := bufio.NewScanner(input)
for scanner.Scan() {
    token := scanner.Text()
}
if err := scanner.Err(); err != nil {
}
```


- Good

```
scanner := bufio.NewScanner(input)
for {
    token, err := scanner.Scan()
    if err != nil {
        return err
    }
}
```

참고문헌

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

- Common Weakness Enumeration, (2022), “CWE CATEGORY: Error Conditions, Return Values, Status Codes”, <https://cwe.mitre.org/data/definitions/389.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

4. Import 블랙리스트

현재, 사용하면 안되는 라이브러리 및 함수를 Import 함으로써 의도치 않은 문제점이 야기될 수 있다.

1. crypto/md5

crypto/md5는 현재 공격이 가능한 취약한 알고리즘이므로, 사용해서는 안된다.

예시

- Bad


```
package main

import (
    "crypto/md5"
    "fmt"
    "os"
)

func main() {
    for _, arg := range os.Args {
        fmt.Printf("%x - %s", md5.Sum([]byte(arg)), arg)
    }
}
```

참고문헌

- OWASP Top 10 Project, (2021), "OWASP Top Ten A02:2021 - Cryptographic Failures", https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- Common Weakness Enumeration, (2022), "CWE-327: Use of a Broken or Risky Cryptographic Algorithm", <https://cwe.mitre.org/data/definitions/327.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

2. crypto/des

crypto/des는 현재 공격이 가능한 취약한 알고리즘이므로, 사용해서는 안된다.

예시


- Bad

```
package main

import (
    "crypto/cipher"
    "crypto/des"
    "crypto/rand"
    "encoding/hex"
    "fmt"
    "io"
)

func main() {
    block, err := des.NewCipher([]byte("sekritz"))
    if err != nil {
        panic(err)
    }

    plaintext := []byte("I CAN HAZ SEKRET MSG PLZ")
    ciphertext := make([]byte, des.BlockSize+len(plaintext))
    iv := ciphertext[:des.BlockSize]
    if _, err := io.ReadFull(rand.Reader, iv); err != nil {
        panic(err)
    }
}
```

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide


```

stream := cipher.NewCFBEncrypter(block, iv)
stream.XORKeyStream(ciphertext[des.BlockSize:], plaintext)
fmt.Println("Secret message is: %s", hex.EncodeToString(ciphertext))
}

```

참고문헌

- OWASP Top 10 Project, (2021), “OWASP Top Ten A02:2021 - Cryptographic Failures”, https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- Common Weakness Enumeration, (2022), “CWE-327: Use of a Broken or Risky Cryptographic Algorithm”, <https://cwe.mitre.org/data/definitions/327.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

3. crypto/rc4

crypto/rc4는 현재 공격이 가능한 취약한 알고리즘이므로, 사용해서는 안된다.

예시

- Bad

```
package main


import (
    "crypto/rc4"
    "encoding/hex"
    "fmt"
)

func main() {
    cipher, err := rc4.NewCipher([]byte("sekritz"))
    if err != nil {
        panic(err)
    }

    plaintext := []byte("I CAN HAZ SEKRET MSG PLZ")
    ciphertext := make([]byte, len(plaintext))
    cipher.XORKeyStream(ciphertext, plaintext)
    fmt.Println("Secret message is: %s", hex.EncodeToString(ciphertext))
}
```

참고문헌

- OWASP Top 10 Project, (2021), "OWASP Top Ten A02:2021 - Cryptographic Failures", https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- Common Weakness Enumeration, (2022), "CWE-327: Use of a Broken or Risky Cryptographic Algorithm", <https://cwe.mitre.org/data/definitions/327.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

4. net/http/cgi

Content-Type 헤더를 명시적으로 설정하지 않으면 net/http/cgi 및 net/http/cgi패키지가 기본적으로 text/html 공격자가 응답 내용의 일부를 제어할 수 있는 경우 Cross-Site Scripting 공격이 가능하다. 따라서 Go 1.15.1 이상의 최신 버전의 GO 사용 하고, net/http/cgi 패키지를 사용하지 않아야 한다.

예시

- Bad


```
package main

import (
    "net/http/cgi"
    "net/http"
)

func main() {
    cgi.Serve(http.FileServer(http.Dir("/usr/share/doc")))
}
```

참고문헌

- OWASP Top 10 Project, (2021), “OWASP Top Ten A03:2021 - Injection”, https://owasp.org/Top10/A03_2021-Injection/
- OWASP Top 10 Project, (2021), “OWASP Top Ten A06:2021 - Vulnerable and Outdated Components”, https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/
- Common Weakness Enumeration, (2022), “CWE-319: Cleartext Transmission of Sensitive Information”, <https://cwe.mitre.org/data/definitions/319.html>
- Common Weakness Enumeration, (2022), “CWE-327: Use of a Broken or Risky Cryptographic Algorithm”, <https://cwe.mitre.org/data/definitions/327.html>

 블록체인연구소 Blockchain Research Institute	스마트 컨트랙트를 위한 Go Language 시큐어 코딩 가이드	
	소속	고려대학교 / 블록체인 연구소
	제목	Go Secure Coding Guide

5. crypto/sha1

crypto/sha1은 현재 공격이 가능한 취약한 알고리즘이므로, 사용해서는 안된다.

예시

- Bad

```
package main

import (
    "crypto/sha1"
    "fmt"
    "os"
)

func main() {
    for _, arg := range os.Args {
        fmt.Printf("%x - %s", sha1.Sum([]byte(arg)), arg)
    }
}
```

참고문헌

- OWASP Top 10 Project, (2021), “OWASP Top Ten A02:2021 - Cryptographic Failures”, https://owasp.org/Top10/A02_2021-Cryptographic_Failures/
- Common Weakness Enumeration, (2022), “CWE-327: Use of a Broken or Risky Cryptographic Algorithm”, <https://cwe.mitre.org/data/definitions/327.html>