SAST常见痛点的CxQL探索

tulliuslin@tencent.com

Agenda

- SAST常见痛点
  - 语言特性覆盖不足
  - 主流开发框架覆盖不足
  - Source/Sink定义太宽泛
  - 预编译识别不准确
  - 控制流支持不足
  - 数据流断裂
  - 逻辑漏洞难以识别
- 使用CxQL自定义规则缓解痛点

Checkmarx

SAST常见痛点

SAST常见痛点

- 语言特性覆盖不足：漏报

1. 动态特性：PHP/JS/Python属于动态特性比较多的语言，source、sanitizer、sink都是可以动态生成的，纯静态分析的SAST无法通过hardcode黑名单的方式进行处理

```php
<?php
    @$_="s"."s"."e"."r";
    @$_="a".$_."t";
    @$_(${"_P"."OS"."T"}[1-2-5]);
?>
```

2. Validation：Go支持在Struct标签定义字段属性

```go
type User struct {
  Id        int       `json:"id"`
  Name      string    `json:"name"`
  Bio       string    `json:"about,omitempty"`
  Active    bool      `json:"active"`
  Admin     bool      `json:"-"`
  CreatedAt time.Time `json:"created_at"`
}
```

SAST常见痛点

- 主流开发框架覆盖不足：漏报

框架名	框架	迭代状态	是否自研	CheckMarx是否支持（默认）
django	python	持续迭代	外部	是
flask	python	持续迭代	外部	是
express	js	持续迭代	外部	是
hapi	js	持续迭代	外部	是
tornado	python	持续迭代	外部	否
lavarel	php	持续迭代	外部	否
thinkphp	php	持续迭代	外部	否
CodeIgniter	php	持续迭代	外部	否
koa	js	持续迭代	外部	否
TypeORM	js	持续迭代	外部	否
eggjs	js	持续迭代	外部	否
gin	go	持续迭代	外部	否
iris	go	持续迭代	外部	否
xorm	go	持续迭代	外部	否
gorm	go	持续迭代	外部	否
beego	go	持续迭代	外部	否

SAST常见痛点

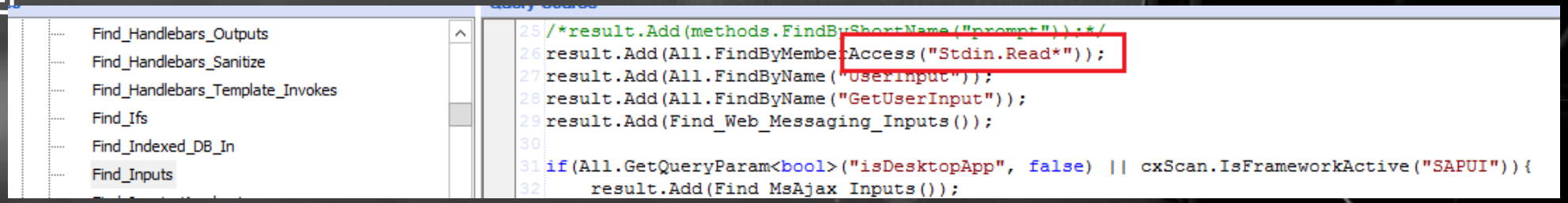
- source/sink定义太宽泛：误报

1. 不是所有header都能作为Source/Sink，也不是当object的attribute被污染时整个object都被污染，然而SAST不做区分

2. Host不可控，导致大量SSRF/任意跳转/DOMXSS误报

```
1.  res.redirect("/login?returnurl=" + req.url); //跳转目的地址需要验证下格式，如果host不可控则忽略，但cx会认为构成漏洞。
```

3. Source不区分入口，例如命令行或web入口，导致大量风险问题变漏洞



SAST常见痛点

- 预编译识别不准确：误报

Go/NodeJS无法识别预编译造成大量误报

```go
    sqlStr := fmt.Sprintf(`insert into %s(cmsid, valid, oper) values(?,1,?) on duplicate key update valid=1, oper=?`,
        getGanyuTable())
    _, err = db.Exec(sqlStr, cmsid, oper, oper)
    return err
```

SAST常见痛点

- 控制流支持不足：误报/漏报
  - 分支判断容易错误，导致数据流错误
  - 无法识别Sanitizer，导致误报

```
134 func getModuleName(r *http.Request) string {
    m
135         m := r.FormValue("m")
    m -> m
136         m = strings.TrimSpace(m)
137         for _, c := range []byte(m) {
138                 if c >= 'a' && c <= 'z' {
139                         continue
140                 }
141                 if c >= 'A' && c <= 'Z' {
142                         continue
143                 }
144                 if c >= '0' && c <= '9' {
145                         continue
146                 }
147
148                 if c == '_' {
149                         continue
150                 }
151                 return ""
152         }
    m -> #0
153         return m
154 }
```

```
51          parseUrl, err := url.Parse(req.RedirectUrl)
52          if err != nil || !(DomainRegex.MatchString(parseUrl.Hostname())) || parseUrl.Hostname() != req.Domain {
53                  log.Errorf("url parse fail. url:%s, domain:%s", req.RedirectUrl, req.Domain)
54                  other.FormatResp(respConstant.CODE_PARSE_URL_FAILED, respConstant.STR_PARSE_URL_FAILED, rsp)
55                  return nil
56          }
```

SAST常见痛点

- 数据流断裂

1. 第三方依赖需要编译时引入，但静态分析无法识别或识别有限
   - Java pom.xml
   - Go go.mod
   - Python requirements.txt
   - …
2. 引擎对语言特性支持存在缺陷

SAST常见痛点

- 逻辑漏洞难以识别

1. 敏感信息泄漏
2. 越权
3. 并发
4. CSRF
5. …

Checkmarx

使用CxQL自定义规则缓解痛点

Checkmarx

使用CxQL自定义规则缓解痛点

痛点	能否使用CxQL缓解	思路
语言特性覆盖不足	否	引擎优化
主流开发框架覆盖不足	是	补充Source/Sanitize/Sink
Source/Sink定义太宽泛	是	按照漏洞风险和利用难度严格限制范围
预编译识别不准确	是	精确计算预编译?占位符以及对应参数位置
控制流支持不足	是	增加对IF、Contains、StartsWith、Endswith、Regrex等校验机制的防护识别
数据流断裂	是	Go主动添加CxGoConfigurationFile.json
逻辑漏洞难以识别	是	匹配字段名和函数名识别敏感数据

主流开发框架适配示例代码

- Go适配Gin

```
// Refer: https://gin-gonic.com/zh-cn/docs/examples/binding-and-validation/

List < string > ginBind = new List<string> {
                "Bind",
                "BindJson",
                "BindXML",
                "BindYAML",
                "BindQuery",
                "ShouldBind",
                "ShouldBindJSON",
                "ShouldBindXML",
                "ShouldBindQuery"
                "ShouldBindYAML",
                "ShouldBindWith",
                "ShouldBindBodyWi
        };

/*
只适用于:
    req := &BlobUploadImgReq{}
    err = ginCtx.BindQuery(req)
*/
CxList variables = Find_UnknownRe
CxList ginCtx = variables.FindByP
CxList bindQuery = ginCtx.GetMemb
CxList req = All.GetParameters(bi
CxList reqStruct = All.FindDefini
result = All.FindByShortName(reqS
```

```
/*

ref: https://www.jianshu.com/p/98965b3ff638
c.String
c.JSON
C.HTML
c.YAML
c.XML
c.ProtoBuf
c.SecureJSON // anti jsonhijacking
c.JSONP
c.AsciiJSON
c.PureJSON
c.Status
c.DataFromReader
c.Redirect

 */

// Find_Handler
CxList contextParam = All.FindByType("gin.Context").FindByType(typeof(ParamDecl));
CxList contextParamRef = All.FindAllReferences(contextParam);
CxList getSource = contextParamRef.GetMembersOfTarget();
//return getSource.GetAssignee();

List<string> methodsWhitelist =
    new List<string> {"String", "HTML", "YAML", "c.XML", "ProtoBuf", "JSONP", "AsciiJSON", "PureJSON", "DataFromReader", "Redirect"};

foreach(CxList Source in getSource){
    if(methodsWhitelist.Contains(Source.GetName())){
        result.Add(getSource.GetAssignee());
    }
}
```

Source/Sink限制示例代码

- Source去除非web入口

```
Find_Deserialization_Inputs
Find_Deserialization_Sanitizers
Find_Deserializers_Constructors
Find_Empty_Strings
Find_Exec_Outputs
Find_File_Open_Refs
Find_File_Read
Find_File_Write
Find_General_Sanitize
Find_HSTS_Configuration_In_Code
Find_HTTP_Response_Write
Find_HTTP_Responses
Find_Imports
Find_Inputs
Find_Integers
```

```
// From user interaction
result.Add(Find_Interactive_Inputs());

//
res

//
res

//
res

//
res
```

```
result = base.Find_Inputs();
// just detect web inputs
result -= Find_Interactive_Inputs();
result -= Find_Console_Inputs();
result -= Find_Read();
result -= Find_Remote_Requests();
```

预编译识别示例代码

- Go精确识别?占位符与对应参数

```
result = base.Find_DB_Sanitize();
CxList methods = Find_Methods();

List<string> sanitizer_methods = new List<string> {"strconv.Atoi", "strconv.ParseInt",
        "*conv.int", "*Int32", "*Int64", "*MysqlRealEscapeString*"};
foreach(string sanitizer_method in sanitizer_methods){
    result.Add(All.FindByName(sanitizer_method) +
        All.FindAllReferences(All.FindDefinition(All.FindByName(sanitizer_method))) +
        All.FindAllReferences(All.FindByName(sanitizer_method)));
}

// general queries
CxList strings = Find_Strings();
CxList decalarators = Find_Declarators();
CxList strAddSanitizers = All.NewCxList();

// possible sql string with place holder ?
// case 1: var := "SELECT c_giturl FROM ABC WHERE c_department=? AND c_center=? and c_team=?"
// case 2: var := "SELECT c_giturl FROM ABC" + "WHERE c_department=? AND c_center=? and c_team=?"
// case 3: sink(ctx, "SELECT c_giturl FROM ABC WHERE c_department=? AND c_center=? and c_team=?")
// case 4: sink( "SELECT c_giturl FROM ABC WHERE c_department=? AND c_center=? and c_team=?")
// case 5: sink(ctx, "SELECT c_giturl FROM ABC WHERE c_department=?" + " AND c_center=? and c_team=?")
// case 6: sink( "SELECT c_giturl FROM ABC WHERE " + "c_department=? AND c_center=? and c_team=?")

CxList strWithQuestionMark = strings.FindByShortName("*?*");
CxList sqlStr = strings.FindByShortNames(
    new List<string> {"*SELECT*", "*INSERT*", "*UPDATE*", "*DELETE*"}, false);

CxList plus = base.Find_BinaryExpr().GetByBinaryOperator(BinaryOperator.Add);

foreach (CxList p in plus) // for strings added with +
{
    try
    {
        BinaryExpr binaryExpr = p.TryGetCSharpGraph<BinaryExpr>();

        if (binaryExpr != null && binaryExpr.Right != null && binaryExpr.Left != null)
        {
            Expression right = binaryExpr.Right;
            CxList cxRight = All.FindById(right.NodeId);
            Expression left = binaryExpr.Left;
            while (left is BinaryExpr) // this is for cases like string1+string2+string3. There're a bit
            loop dose
            {
                if (left != null && ((BinaryExpr) left).Right != null)
                {
                    Expression leftRight = ((BinaryExpr) left).Right;
                    CxList cxLeftRight = All.FindById(leftRight.NodeId);
                    if ((sqlStr * cxLeftRight).Count > 0)
                    {
```

控制流识别示例代码

- 识别IF、Contains、Startswith、Endswith等防护逻辑

```
CxList isAbs = All.FindByMemberAccess("path.IsAbs");
CxList isBase = All.FindByMemberAccess("path/filepath.*").FindByShortName("Base");
CxList isNotAbs = isAbs.GetFathers().FindByType(typeof(UnaryExpr)).FindByShortName("Not");

// If Statments that includes isAbs function
CxList ifsWithIsAbsStmt = isAbs.GetAncOfType(typeof(IfStmt));
CxList ifsWithIsNotAbsStmt = isNotAbs.GetAncOfType(typeof(IfStmt));
ifsWithIsAbsStmt = ifsWithIsAbsStmt - ifsWithIsNotAbsStmt;

// All IF blocks containing the sanitizers
CxList ifBlocksWithIsAbs = allSanitizers.GetByAncs(ifsWithIsAbsStmt);
CxList ifBlocksWithIsNotAbs = allSanitizers.GetByAncs(ifsWithIsNotAbsStmt);

// Reduce sanitizers scope to places where the input path is relative and not absolute
CxList ifsBlocksSanitized = All.NewCxList();
CxList fBlocks = All.NewCxList();
CxList tBlocks = All.NewCxList();
fBlocks.Add(All.GetBlocksOfIfStatements(false));
tBlocks.Add(All.GetBlocksOfIfStatements(true));
ifsBlocksSanitized.Add(ifBlocksWithIsAbs.GetByAncs(fBlocks));
ifsBlocksSanitized.Add(ifBlocksWithIsNotAbs.GetByAncs(tBlocks));
ifsBlocksSanitized.Add(isBase);
```

数据流断裂粘合示例代码

- 识别Go.mod中当前仓库的地址，从而识别同一个仓库的绝对路径引用：

在该项目下添加一个配置文件 CxGoConfigurationFile.json，可让checkmarx知道文件1中引用的“git.code.abc.com/project/abc/logic”跟自己是在同一仓库根目录下的“logic目录“

格式：

{
“Package”:“git.code.abc.com/project/abc/logic"
}

- 使用CustomFlow

```
// CustomFlows_Inputs
// Build flows between the definition of a input variable and its references
// for example: for source code $a = $_GET[x];  build all flows between $a and its references

CxList inputs = Find_Interactive_Inputs();
CxList inputsFathers = inputs.GetFathers();

CxList inputsDecls = All.GetByAncs(inputsFathers).FindByAssignmentSide(CxList.AssignmentSide.Left);

CxList inputsRefs = All.FindAllReferences(inputsDecls).FindByType(typeof(UnknownReference)) - inputsDecls;

foreach(CxList input in inputsRefs)
{
    CustomFlows.AddFlow(inputsDecls, input);
}
```

逻辑漏洞识别示例代码

- 根据关键字识别敏感信息，以越权漏洞为例，最常见的是表主键自增且被作为查询参数，通过遍历主键批量拉取敏感信息，同时常见防护逻辑为加上当前用户作为限制条件，因此可自定义规则：

```
CxList tables = All.FindByShortName("*orders*", false);
tables.Add(All.FindByShortName("*credit*", false));
tables.Add(All.FindByShortName("*invoice*", false));
tables.Add(All.FindByShortName("*booking*", false));
tables.Add(All.FindByShortName("*bill*", false));
tables.Add(All.FindByShortName("*payment*", false));
tables.Add(All.FindByShortName("*account*", false));
tables.Add(All.FindByShortName("*cash*", false));
tables.Add(All.FindByShortName("*customer*", false));

CxList inputs = Find_Interactive_Inputs();
CxList db = Find_DB_In();

CxList user = All.FindByShortName("*user*", false);
user.Add(All.FindByShortName("*cust*", false));
user.Add(All.FindByShortName("*member*", false));

db = db.DataInfluencedBy(tables);
db -= db.DataInfluencedBy(user);
CxList sanitize = Find_Parameter_Tampering_Sanitize();

result = inputs.InfluencingOnAndNotSanitized(db, sanitize);
```

```
result = All.FindByMemberAccess("AccessReferenceMap.getDirectReference");

//Find data structure get methods
CxList getMethod = All.FindByMemberAccess(".get");
CxList dataStractureGet = getMethod.FindByMemberAccess("Attributes.get");
dataStractureGet.Add(getMethod.FindByMemberAccess("Collection.get"));
dataStractureGet.Add(getMethod.FindByMemberAccess("List.get"));
dataStractureGet.Add(getMethod.FindByMemberAccess("Map.get"));
dataStractureGet.Add(getMethod.FindByMemberAccess("Table.get"));
dataStractureGet.Add(getMethod.FindByMemberAccess("Vector.get"));
result.Add(dataStractureGet);

// Find conditions variables of if statements
CxList ifStmt = base.Find_Ifs();
CxList conditions = All.NewCxList();
foreach (CxList singleIf in ifStmt)
{
    try
    {
        IfStmt stmt = singleIf.TryGetCSharpGraph<IfStmt>();
        if (stmt.Condition != null)
        {
            conditions.Add(stmt.Condition.NodeId, stmt.Condition);
        }
    }
    catch (Exception ex)
    {
        cxLog.WriteDebugMessage(ex);
    }
}

CxList conditionsVars = All.GetByAncs(conditions);
result.Add(All.FindAllReferences(conditionsVars));
```

逻辑漏洞识别示例代码

- 也可以通过识别切面或者统一防护函数进行判断

1. Django/ Koa/Gin中的middleware
2. Java中的Filter

```
Find_DB_Out_Sybase
Find_Django
Find_Django_Config
Find_Django_Inputs
Find_Django_Model_Components
Find_Django_Outputs
Find_Django_XSRF_Sanitize
Find_Empty_Strings
Find_Encrypt
Find_Evil_Strings
Find_Exception_Information
Find_Header_Outputs
Find_HSTS_Configuration_In_Code
Find_HTTP_Response_Write
Find_HTTP_Responses
Find_HTTPServer_Inputs
Find_HTTPServer_Log
Find_HTTPServer_Outputs
Find_Imports
```

```
/*
    This Query Finds the Sanitizers for Django
    which are protected by the CsrfMiddleWare
    It finds the Middleware by decorators, and
    by application Dependency Injection
    It uses the Settings > Routings > Controllers
    Flow To search for sanitized XSRF
*/
if (Find_Django().Count != 0)
{
    CxList strings = Find_Strings();

    CxList DomainSafetyOrigin = strings.FindByShortName("*django.middleware.csrf.CsrfViewMiddleware*");

    // Detect Django CsrfMiddleWare
    CxList DomainSafety = DomainSafetyOrigin.GetAncOfType(typeof(Declarator))
            .FindByShortName("MIDDLEWARE_CLASSES");

    DomainSafety.Add(All.FindByShortName("MIDDLEWARE_CLASSES")
            .GetByAncs(DomainSafetyOrigin.GetAncOfType(typeof(AssignExpr))));

    // Find Affected Applications
    CxList ProtectedFiles = All.NewCxList();
    foreach(CxList Middleware in DomainSafety){
        CSharpGraph midWare = Middleware.GetFirstGraph();
```

/ Thank you

www.checkmarx.com