一种可扩展 Web 模型安全机制①

何 旭

(达州职业技术学院 机电与信息工程系, 达州 635001)

摘 要：Web 扩展对于安全的一贯性具有综合影响。定义了影响安全模型的 Web 入侵、网络入侵与 Gadget 入侵的行为能力。提出了 Web 扩展模型具有 Web 概念的浏览器、服务器、协议的类、属性与方法。使用该扩展模型研究了重定向请求到入侵者服务器的交叉区域资源共享安全机制，以及相同区域源网站的引用验证安全机制，分析了两个机制的脆弱性，并提出了忽略重定向请求与禁止出口引用的解决方案。

关键词：重定向; 交叉区域; 资源共享; 引用验证; 安全机制

Extensible Web Model Security Mechanism

HE Xu

(Department of Mechatronics and Information Engineering, Dazhou Vocational Technology College, Dazhou 635001, China)

Abstract：Web extensions for security consistency have a comprehensive impact. It defines the effect of security model capacity in the Web Attacker, Network Attacker and Gadget Attacker. Web extension model proposed the concept of a Web browser, server, protocol type, properties and methods. Finally, it studied using the extended model intruder server redirects the request to cross-origin resource sharing security mechanisms, and the same-origin site refer validation security mechanism, analysis of the vulnerability of the two mechanisms, and proposes ignore redirect request and suppress all outgoing refer solutions.

Key words：redirect; cross-origin; resource sharing; refer validation; security mechanism

1 引言

Web 具有多方面安全需求的复杂分布式应用，Web 包括了被认为具有现存安全性的大量 Web 应用。大多数 Web 浏览器、服务器、网络协议、浏览器扩展及其安全机制在设计时不具有安全分析基础。更为复杂的是，Web 正不断地快速包括并引入一些新的浏览器特征、协议与标准。这些包括与引入的性能通常是复杂的，同时也可能引入一些新威胁和破坏 Web 原来一贯的安全性[1,2]。

尽管 Web 应用可以使用交叉区域的 GET 与 POST 请求，由于浏览器保护有些属性，这样就可以认为在有些 Web 应用中浏览器根本不使用 DELETE 方法产生交叉区域的 HTTP 请求。在 Web 新引入因素的安全分析时，就必须检查新因素的安全一贯性，诱发引起现存 Web 应用的脆弱性。会话完整性可以通过记录产生每个 HTTP 请求的原因(HTTP 请求是由脚本还是 HTTP 重定向调用 API 产生)，并且可以检查入侵者是否在原因链上。一般地，当服务器执行基于 HTTP 请求的事务时，服务器通常认为请求是由可信任网站产生而不是由入侵者产生的。

对于网络新协议安全性的验证与扩展模型的检查可以使用工具、基于限制以及证明等方法[3,4]，也可以使用正式验证 Web 服务安全[5]。

2 入侵威胁

2.1 Web 入侵

Web 入侵者至少能够控制一个 Web 服务器，同时还能够使用任意应答内容响应 HTTP 请求。同时，如

① 收稿时间:2011-12-03;收到修改稿时间:2012-02-14

果 Web 入侵者具有“根访问”权限访问 Web 服务器，Web 入侵者就能够通过控制大量 DNS 服务器指向任何类型服务器。也就是说，DNS 域名如果能够指向入侵者的 Web 服务器，入侵者就能获得并拥有由可信任的认证机构认证的 HTTPS 认证，使用这些认证 Web 入侵者能够在 URL 中注入恶意内容。

Web 入侵者既不能从接收端偷听信息也不能在发送端伪造信息，入侵者没有专门的网络特权，但是入侵者能够响应 HTTP 请求指向自己的服务器，并且入侵者能够从其控制的网络节点发送 HTTP 请求给可信服务器，这些 HTTP 请求不必遵守 HTTP 协议，也不需要入侵者处理该响应。

当用户访问入侵者的网站时，入侵者就被导入用户浏览器。一旦导入，入侵者就可以访问用户的受浏览器安全策略限制的 API。例，通过调用函数 window.open() 创建浏览器新窗口。浏览器 API 通过超链接或 HTML 表单产生交叉区域 HTTPS 请求，因为这些请求包括了用户 Cookies 以及浏览器解释的应答，所以入侵者可以使用这些带参数的 API 直接发送 HTTP 请求[6]。

2.2 主动网络入侵

主动网络入侵者既拥有 Web 入侵者的所有能力，又能够读取、控制、阻止没有加密的网络传输信息。虽然可信认证授权可以发布入侵者的 HTTPS 认证给恶意 DNS 域名，但是可信认证授权不可以发布入侵者的认证给可信 DNS 域名，这样入侵者就不能够破坏可信 HTTPS 传输代理。如果无正确的认证，入侵者就不能读取或修改 HTTPS 请求以及应答的内容。

2.3 Gadget 入侵

Gadget 入侵者既具有 Web 入侵者的所有能力，又能够依靠 Web 应用插入一些受限制的内容到可信网站。例，插入超链接到电子邮件或博客文档、插入图像到论坛以及在高级应用中插入插件等。

3 Web扩展模型形式化描述

扩展的安全模型中定义了新类 Principal、NetworkEndpoint 与 NetworkEvent 等，类 NetworkEvent 表示具有来自 NetworkEndpoint 的发送端与到 NetworkEndpoint 接收端的类，且是 Event 的子类。NetworkEvent 既可以看作是 HTTPRequest 也可以看作是 HTTPResponse，它包括 Method 方法、HTTPRequestHeader 与 HTTPResponseHeader 的 HTTP 具体说明信息。类图如图 1 所示，虚线方框是可扩展的类，该模型图提供了模型形式化的概念关系[7]。这些类扩展后形式化描述分别如下所示：

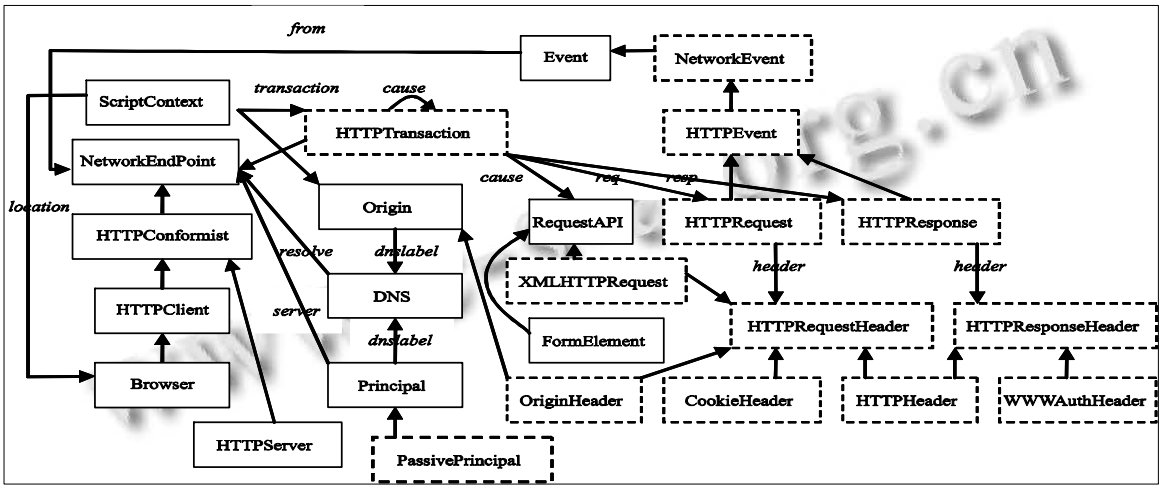

图 1　Web 安全扩展模型图

NetworkEvent 扩展 Event 事件，审核各网络端节点事件。

```
abstract sig NetworkEvent extends Event {
  from：NetworkEndpoint,
  to：NetworkEndpoint
}
```

HTTPEvent 扩展 NetworkEvent 事件，审核区域源的引用。

```
abstract sig HTTPEvent extends NetworkEvent {
  host：Origin
```

}

HTTPRequest 扩展 HTTPEvent 事件，跟踪路由以及设置请求头。

```
sig HTTPRequest extends HTTPEvent {
  method：Method，path：Path，
  headers：set HTTPRequestHeader
}
```

HTTPResponse 扩展 HTTPEvent 事件，记录状态以及设置请求头。

```
sig HTTPResponse extends HTTPEvent {
  statusCode：Status,
  headers：set HTTPResponseHeader
}
```

3.1 类 Principal

Principal 是一个实体类，控制 NetworkEndpoint 以及 DNSLabel 方法，从而用户完全有权限代表主机名。

```
abstract sig Principal {
  servers：set NetworkEndpoint,
  dnslabels：set DNS
}
```

模型包括了类 Principal 的子类层次结构，各层次的每一级使用一些限制，并且通过添加参数说明如何与其它对象相互作用。在实例中属于实体类 Principal 的服务器遵循网络拓扑结构定义的 HTTP 路由规则。

```
abstract sig PassivePrincipal
  extends Principal{} {
  servers in HTTPConformist
}
```

3.2 类 Browser

类 Browser 是 HTTPClient 的子类，类 HTTPClient 具有可信的 CertificateAuthority 集与 ScriptContext 集，也可以将 Browser 当作 ScriptContext 的一个属性。

```
abstract sig Browser
  extends HTTPClient {
  trustedCA：set CertificateAuthority
}
sig ScriptContext {
  owner：Origin，location：Browser,
  transactions：set HTTPTransaction
}
```

浏览器使用已验证 CA 设置来查验 HTTPS 认证，其认证过程是通过 NetworkEndpoint 响应 HTTPS 请求完成。ScriptContext 也有类 Origin 和 HTTPTransaction，类 Origin 通过浏览器 API 实现相同区域策略。RequestAPI 的子类 XMLHTTPRequest 的作用是预防由外部区域发起具有 ScriptContext 内容的 HTTPRequest 请求，ScriptContext 的属性 transaction 是由 Script Context 方法产生的 HTTPTransaction。

3.3 方法 cause

类 HTTPTransaction 有一个 cause 方法，它既有 HTTPTransaction 又有 RequestAPI。每个 RequestAPI 强制执行由 API 产生的 HTTPRequest 约束机制，约束 FormElement 产生 GET 和 POST 请求。

```
fact {
  all t:ScriptContext.transactions |t.cause in FormElement implies  t.req.method in GET + POST
}
```

使用方法 cause 能够限制类 Principal 参与产生 HTTPTransaction。

```
fun involvedServers[t:HTTPTransaction]:set NetworkEndpoint{
  (t.*cause & HTTPTransaction).resp.from+ getTransactionOwner[t].servers
}
pred webAttackerInCausalChain[t:HTTP Transaction]{
  some (WEBATTACKER.servers& involved Servers[t])
}
```

4 交叉区域资源共享安全分析

交叉区域资源共享（CORS）使网站有选择地退出浏览器安全保护机制[8]。特别地，返回不同 HTTP 头，网站能够使浏览器与特定区域一起共享 HTTP 应答的内容[9]，或者让特定区域发出其它禁止请求。CORS 能够区分简单请求与复杂请求，在简单请求中发送交叉区域是安全的，然而在复杂请求中发送具有潜在危险请求前需要发送 pre-flight 请求用以征询服务器允许。如果 Pre-flight 请求没有被适当处理，复杂请求就可能受到安全威胁。

4.1 建模原理

使用 HTTPRequest 构建其子类 PreflightRequest，并通过浏览器使用特殊语义产生。尽管浏览器重用 XMLHTTPRequest 的 Java 脚本 API 实现 CORS，但是当前与过去行为对比是容易的，使用 CORS 所包含的 HTTP 头构建 CORSResponseHeader 子类。同样，使用 NetworkEndPoint 可以构建在 HTTP 响应中根本不包括任何 CORS 头的服务器。如图 1 所示。

```
fact {
  all p:PreFlightRequest | {
    p.method = OPTIONS and
    some p.headers& AccessControlRequestMethod and
    some p.headers & OriginHeader and
    some p.headers & Access Control Request eaders
  }
}
fact {
  all t:HTTPTransaction,sc:ScriptContext |{
    t in sc.transactions and t.?cause in
    (XMLHTTPRequest2+HTTPTransaction)
} implies {
  isPreFlightRequestTransaction[t]
  or isSimpleCORSTransaction[t]
  or isComplexCORSTransaction[t]
  or (not isCrossOriginRequest[t.req])}
}
```

4.2 脆弱性

由于传统服务器可以重定向请求到入侵者服务器，所以该模型可能产生破坏 Web 安全固定性的情况。浏览器跟踪这些重定向，可以让入侵者服务器返回 CORS 头，CORS 头有选择地使服务器接收不同类型的请求。在实际应用中，由于 Web 站点包含开放重定向，所以入侵是存在的，但是构建模型并不显式包含重定向。由于模型并没有禁止这些响应，服务器可以重定向这些请求到任意地址。

4.3 解决方案

针对上述分析，比较有效的解决方案是忽略重定向请求，可以通过添加下列语句修改协议的安全性。

```
fact {
  all first:HTTPTransaction | {
    first.req in PreFlightRequest and first.resp. status Code in RedirectionStatus
  } implies no second:HTTPTransaction |second.cause = first
}
```

5 引用验证安全分析

通过验证引用，Web 网站就能够预防交叉请求伪造(CSRF)和交叉脚本(XSS)。当用户请求来自目标网站的网页，并且该网站通过跟踪来自其它相同输入网站的链接时入侵就会产生，除非引用源自相同区域的网站或者请求指向入口页面，为了预防这类入侵，网站拒绝 HTTP 请求，并且认真检查 CSRF 与 XSS 的脆弱性[10,11]。如果不检查脆弱性，就只有使用引用来自相同区域的网站，或者请求重定向到入口网页。如图 2 所示，图中虚线表示入侵引用链接路径。

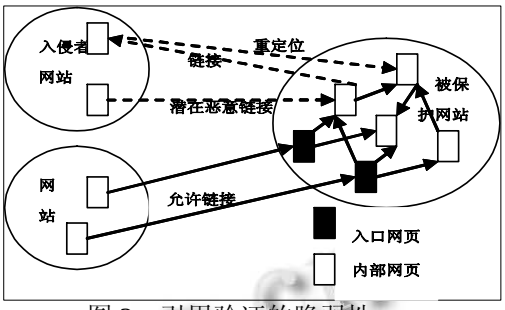

图 2　引用验证的脆弱性

5.1 建模原理

通过使用安全模型的引用部分，首先添加一个用于显示请求行为的新类 ReferProtected，然后再添加一个只用于允许具有外部引用的“LOGIN”网页 HTTP 请求约束。

```
fact {
  all aReq:HTTPRequest | {
    (getTransaction[aReq].resp.from in Refer Protected。servers )
    and is CrossOrigin[aReq]
  } implies aReq.path = LOGIN
}
```

5.2 脆弱性

如果网站首先发送请求到入侵者网站，入侵者就可以通过使用 CSRF 登录入侵到被保护的网站（如图

2 虚线所示)。也就是说，如果入侵者能够插入超链接到可信任网站，用户就可以跟踪该链接，在引用中使用可信网站的URL产生一个HTTP请求链接到入侵者的网站，这样入侵者就能够发出重定位请求链接到可信任网站。

5.3 解决方案

由于引用已经广泛地应用到各类网站，所以通过网页来修改引用验证的脆弱性是比较困难的，一个可行的解决方案是网站禁止使用所有出口引用。

6 仿真实验与结果分析

研究实现了一个 Web 扩展模型仿真实验。仿真使用安全的常量作为条件，要求能够找出一个反例条件，绑定搜索一个有限大小的所有应用顶层的验证，并且可以允许指定类型细粒度的界限功能。

对于交叉区域资源共享与引用验证这两个案例的研究，仿真实验发现漏洞数量和时间测量分析产生的合取范式(CNF)及解析时间，找到了每个案例的一个解决方案，见表 1 所示。实验观察到插入一些新代码、子句与 CNF 产生和解析时间没有明显的相关性，所以本仿真也可以作为其它常用应用解决方案。

表 1 两个实验案例的数据表

案例名称	随机插入代码(行)	子句	CNF 产生时间(秒)	CNF 解析时间(秒)
交叉区域资源	78	9778	23.91	81.67
引用验证	39	9782	30.75	69.02

仿真实验能够在很短的时间找到一个存在的反例，如果规模增加则所用时间会成倍增加。为了量化该案例，实验分析引用验证安全的仿真入侵前后情况，见图 3 所示。

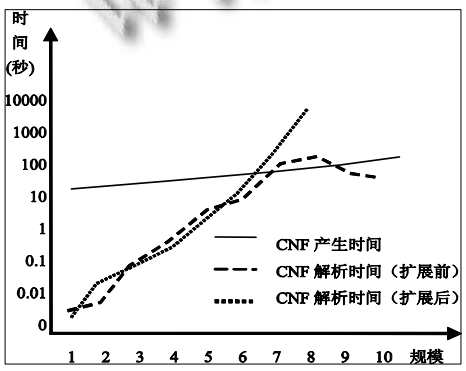

图 3 扩展前后 CNF 产生与解析时间的对比图

7 结语

扩展 Web 安全模型包括 Web 安全机制的浏览器、服务器、HTTP、Cookies、脚本内容以及其安全属性的扩展。安全模型的 Web 入侵、网络入侵到 Gadget 入侵具有递进增加的威胁性。使用安全扩展模型可以分析检查的大多数 Web 机制的安全性，Web 平台众多不同插件交互作用增加了网络的复杂性与脆弱性。

参考文献

1 陈天平,许世军,张串绒,等.基于攻击检测的网络安全风险评估方法.计算机科学,2010,37(9):94–96.

2 Van Kesteren A. Cross-origin resource sharing. http://dev.w3.org/2009/waf/access-control,2009.

3 Mitchell JC, Roy A. Protocol Composition Logic. Electronic Notes in Theoretical Computer Science, 2007,172:311–358.

4 Barth A, Jackson C, Mitchell J. Securing frame communication in browser. Proceedings of the 17th conference on Security symposium. USENIX Association,2008.17–30.

5 Gordon A, Pucella R. Validating a web service security abstraction by typing. Formal Aspects of Computing, 2005,17(3):277–318.

6 Klose T. Confused deputy attack on cors. http://lists.w3.org/Archives/Public/public-webapps/2009AprJun/1324.html, 2009.

7 Akhawe D, Barth A, Lam PE. Web security model implementation. http://code.google.com/p/websecmodel,2010.

8 Barth A, Caballero J, Song D. Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. Proceedings of the 2009 30th IEEE Symposium on Security and Privacy:360–371.

9 Stamm S. Content security policy. https://wiki.mozilla.org/Security/CSP/Spec, 2009.

10 Magazinius J, Askarov A, Sabelfeld A. A lattice-based approach to mashup security. The 5th ACM Symposium on Information, Computer and Communications Security. ACM,2010.

11 Barth A, Jackson C, Hickson I. The http origin header. http://tools.ietf.org/html/draft-abarth-origin,2010.