“基于主机”的应用层入侵阻断系统

Fabrice A. Marie – 方政信
fabrice.marie@fma-rms.com

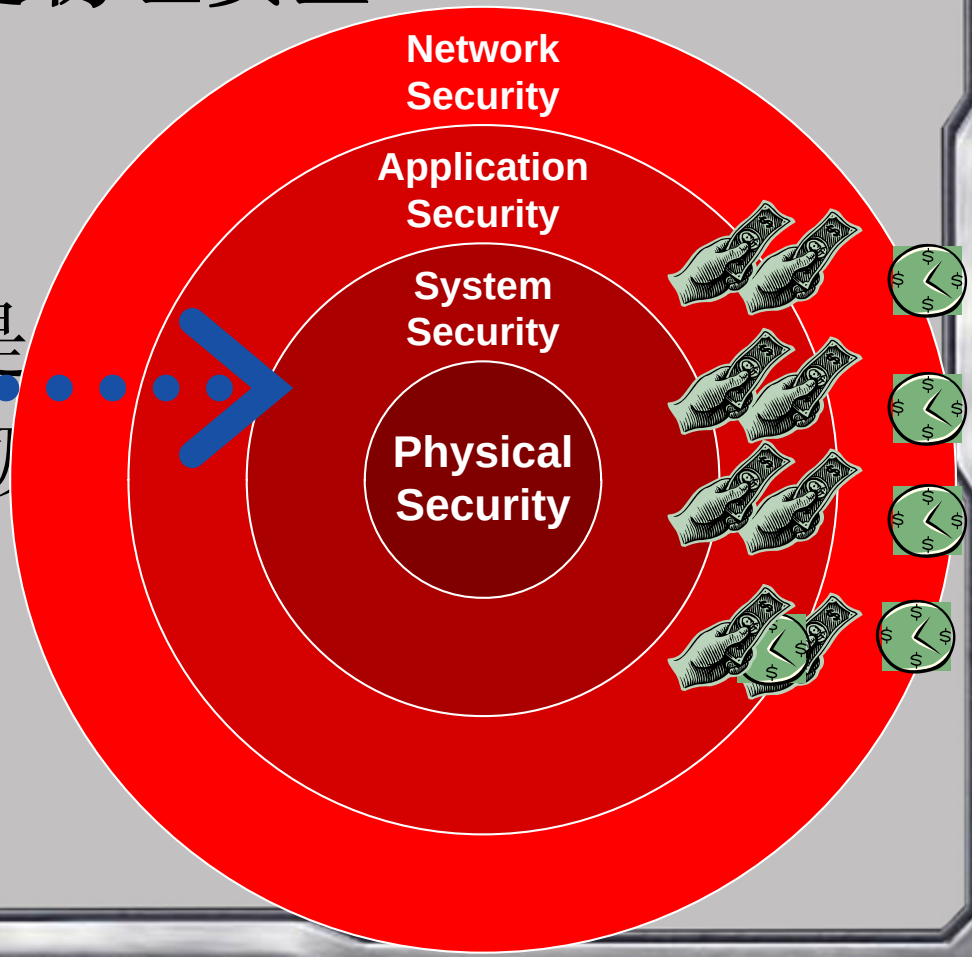
X’Con2006

内容列表

- 为什么攻击Web应用?
- 为什么需要检测?
- 当前IDS技术存在的问题
- 恶意修改的检测
- 攻击行为的检测
- HAIDS → HAIPS
- Framework
- 一些缺陷的说明
- 结论

应用层安全的重要性

- 最重要的当然是物理安全!
- 应用层安全只是
  第3层的优先级



为什么攻击Web应用?

- 简单答案: 因为容易得逞!
- 应用本身没有得到应有的重视....
  - 你为什么需要网络?
  - 你为什么需要工作站?
    - ... 是为了运行应用软件!
- 应用层攻击是非常高效的
  - 网络层攻击是缓慢而痛苦的。攻击者需要:
    - 突破2-3层防火墙
    - 渗透入系统
    - 提升在系统中的权限
    - 找到执行欺骗操作的渠道
  - 而应用层的攻击则要快速得多:
    - 无需突破防火墙
    - 无需渗透入系统
    - 只需专注于应用软件!

为什么攻击Web应用? (继续)

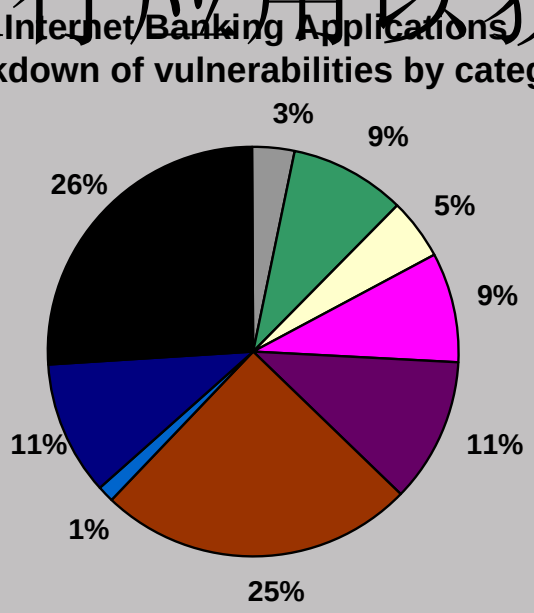
- 应用层攻击一般来说比较简单
  - 如果不简单，那么对应的网络层攻击则更是麻烦！
- 应用层领域技术的缺乏
  - 开发者/架构师/程序员都缺乏技巧
- 你对网络有完全的控制，但对应用却不是
  - 网络使用标准的组件
  - 应用是一个单一的软件

- 因为在别人工作中找问题很有趣
- 因为你能在从中获取收益！
  (并且，犯罪的目标总是金钱)

攻击网络银行应用以获取利益

- 我们普遍能发现的对网络银行应用的欺诈:
  - 读取其他客户的帐单
  - 读取其他客户的个人信息
    - 对更进一步的攻击很有用
      - 身份窃取
  - 读取其他客户的账户操作消息
  - 通过各种渠道窃取金钱
    - 直接转账及其他
  - 以折扣价买进股票
  - 逃避交易费用
  - 各种针对支付网关的回放攻击
  - 破坏交易记录
  - 修改其他客户的个人细节
    - 对更进一步的攻击很有用
      - 用户冒充

攻击网络银行应用以获取利益 (继续)



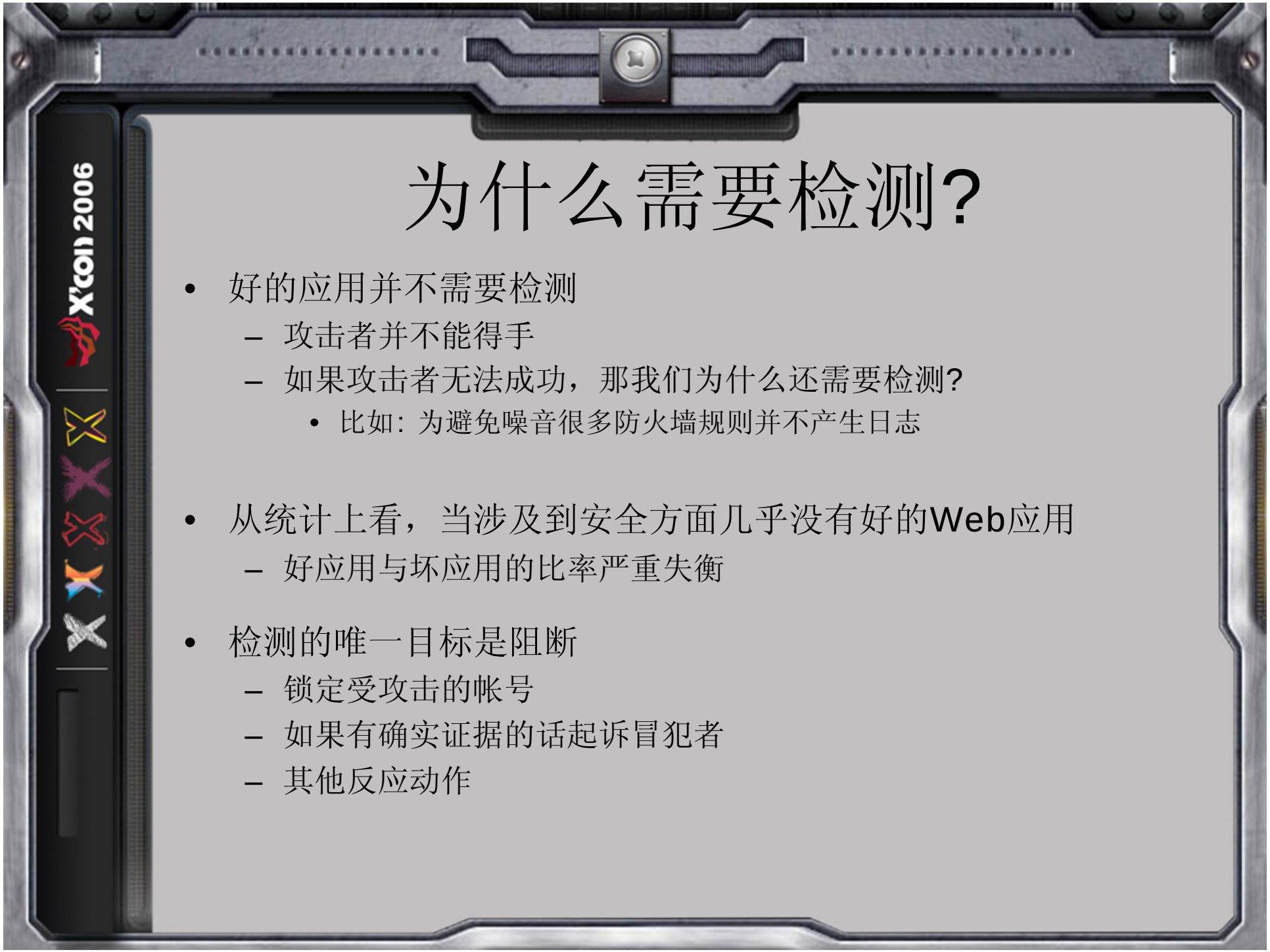
Last 17 internet banking applications we audited

Applications we could steal money from: 100%

Applications we could steal personal information from: 100%

275 vulnerabilities
429 beta scripts
341 unnecessary files

average: **16 vulnerabilities** per application



为什么需要检测?

- 好的应用并不需要检测
  - 攻击者并不能得手
  - 如果攻击者无法成功，那我们为什么还需要检测?
    - 比如: 为避免噪音很多防火墙规则并不产生日志
- 从统计上看，当涉及到安全方面几乎没有好的Web应用
  - 好应用与坏应用的比率严重失衡
- 检测的唯一目标是阻断
  - 锁定受攻击的帐号
  - 如果有确实证据的话起诉冒犯者
  - 其他反应动作

为什么需要检测? (继续)

- 为什么甚至要在攻击者得手之前起诉他们?
  - 很少有攻击者因为网络层的证据被起诉
    - 因为TCP/IP协议的简单性或者说复杂性
      - 端口扫描，PING探测，ARP映射及其他手段
      - 有时候甚至无法从正常情况下筛选出来

    → 证据在法庭上不受支持
  - 应用层的证据则可信多了
    - 恶意修改过的数据流可以被检测到
    - 明确的故意行为

    → 证据在法庭上受支持

- 当前的安全应用可阻止攻击(非常少)
  - 采用严格的验证
  - 采用严格的逻辑控制和流控制
  - 但它们只是在处理错误而不是攻击
    - 它们不能阻止进一步的攻击：没有动作

前IDS技术存在的问题

- 几乎是标记出攻击的唯一技术
- 在能执行检测之前必须先定义特征...
  - 典型的网络攻击存在可检测的迹象
    - 反向的root shell，协议中的可疑字串，其他异常
  - 典型的攻击存在基于主机的IDS可检测的迹象
    - 文件的修改，可疑的日志条目或其他异常
- 如何定义一个应用层攻击或滥用？

当前IDS技术存在的问题 (继续)

- 如何定义一个应用层攻击或滥用？
  - 可以运用同样的手段检测典型的攻击
    - SQL注入
    - 跨站脚本攻击
    - 用户名/口令的暴力猜解
    - 缓冲区溢出
    - [广告]
      更多细节可见我在HITB talk in Kuala Lumpur
- IDS如何了解一个逻辑错误？
  - 例如.: IDS不知道银行帐号
    → 它不可能知道我是从别人的帐号而不是从自己的帐号上转出钱

当前IDS技术存在的问题 (继续)

- 于网络的应用层攻击检测系统(Network-based Application Intrusion Detection Systems，NAIDS)
  - 需要普适化以监视所有的Web应用
  - 既然这样也就只能检测一般的攻击
    - SQL注入，跨站脚本，缓冲区溢出，暴力破解

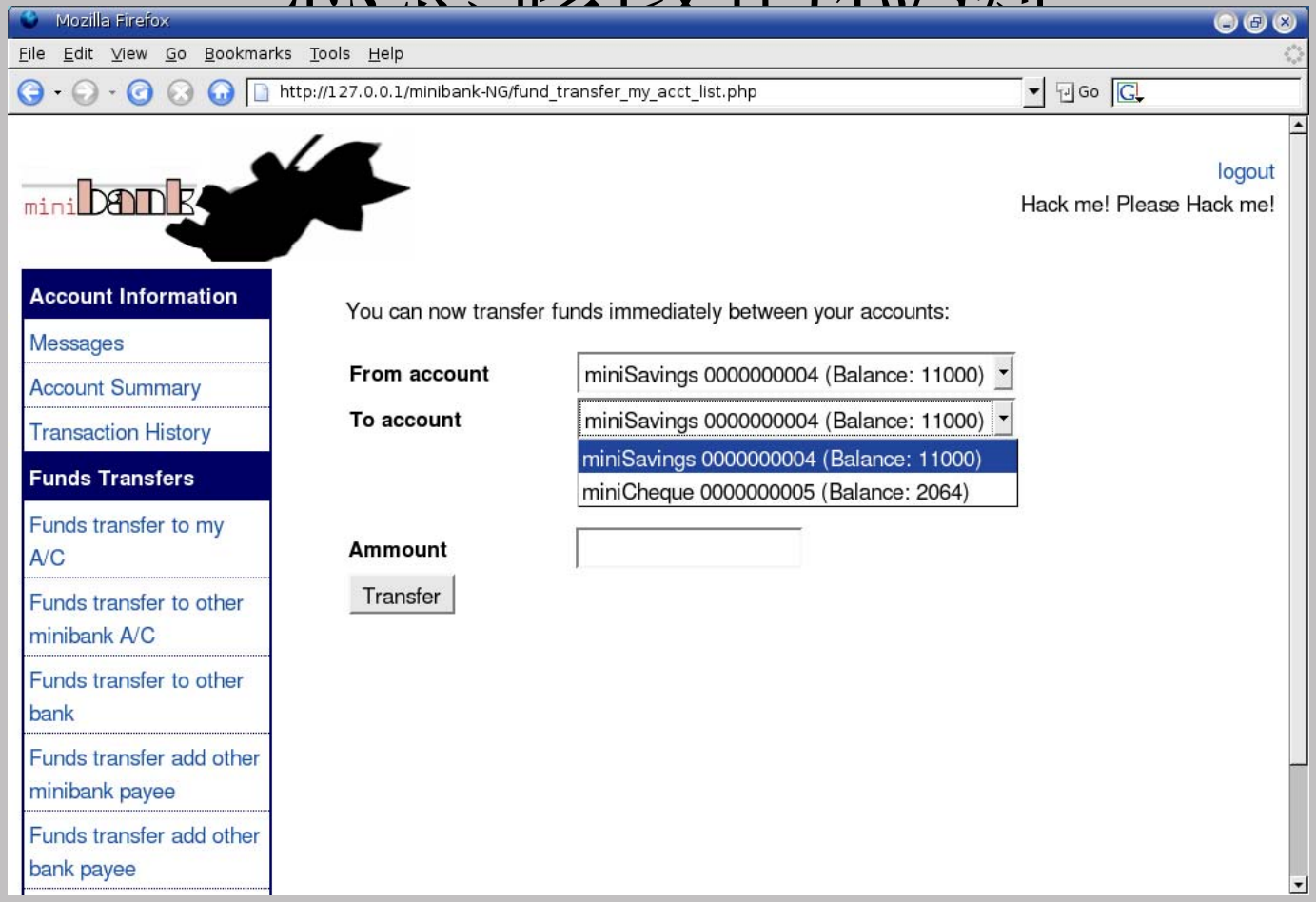
- 为了检测到更多的攻击，我们需要基于主机的应用层入侵检测系统(Host-based Application Intrusion Detection System，HAIDS)
  - 采用特殊的框架形式植入应用软件
  - 完全了解应用软件，包括它的参数和业务逻辑
  - 知道什么只是一个错误而什么却是一次攻击

恶意修改的检测

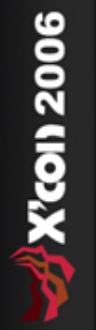
- Web应用软件采用对话框与用户交互：
  - 按钮和复选框
  - 字段值
  - 隐藏字段值
  - 下拉列表
  - 选项列表
  - 更多的交互元素...

- 有些是自由表格
  - 比如.:用户可以随意地填入字串
- 有些是受限的 (理应是这样的)
  - 比如: 下拉列表限制了用户的选择

恶意修改的检测 (继续)

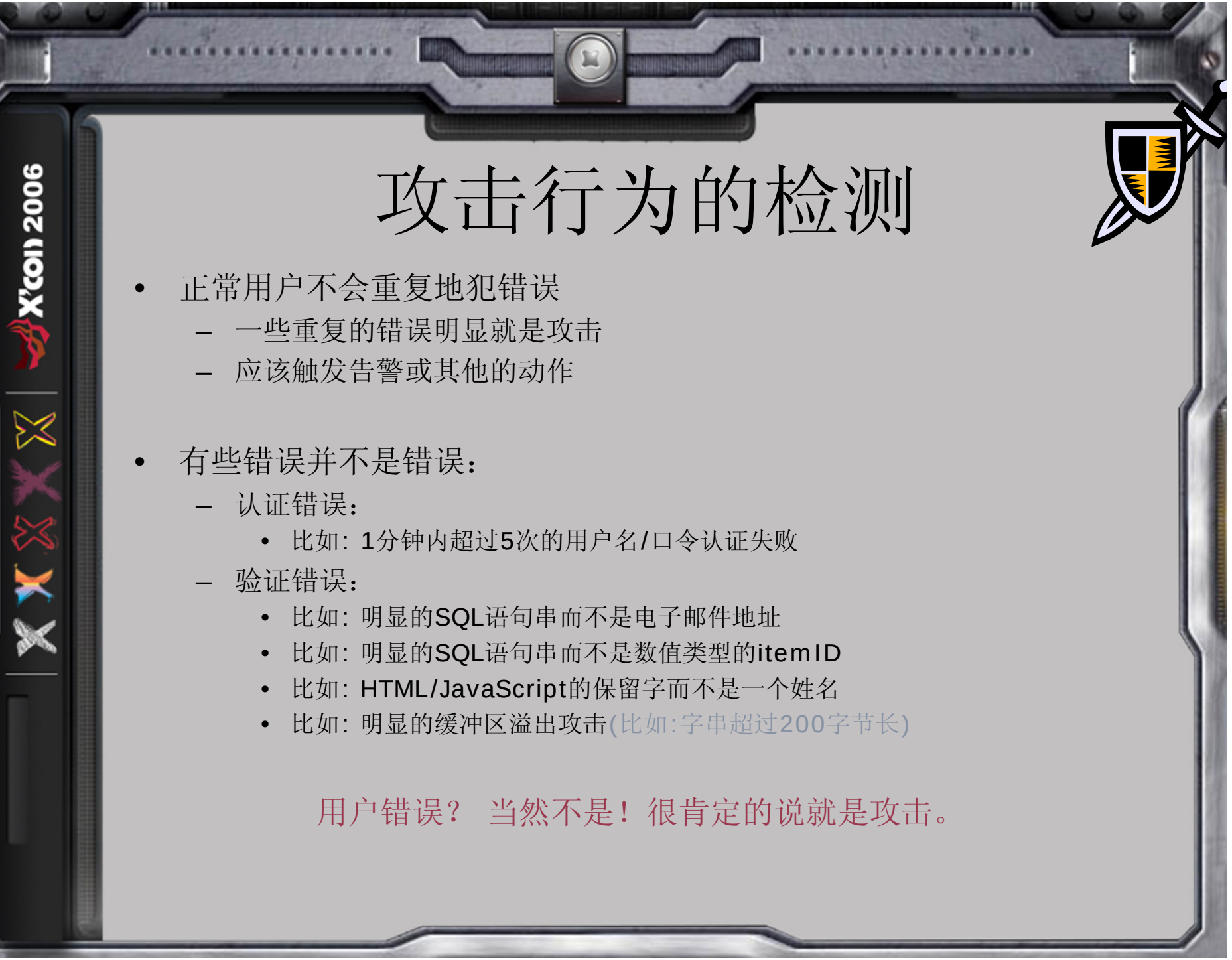


- "to account" 是受限参数
- "amount" 是自由字段



恶意修改的检测 (继续)

- 受限参数不应该被用户修改 (理应如此)
  - 下拉列表 (`<select><option>...`)
  - 按钮 (`<input type="radio" ...`)
  - 复选框 (`<input type="checkbox" ...`)
  - 隐藏字段 (`<input type="hidden" ...`)
  - 固定长度的文本输入框 (`<input maxlength="10" ...`)
  - cookies
- 所以只有攻击者才会修改它们(使用代理)
  - 如果你修改了这样的参数，你就是可疑的
  - 服务器在发送给客户端之前设置了这些值
    - 所以服务端可以轻易地检测到字段的改变

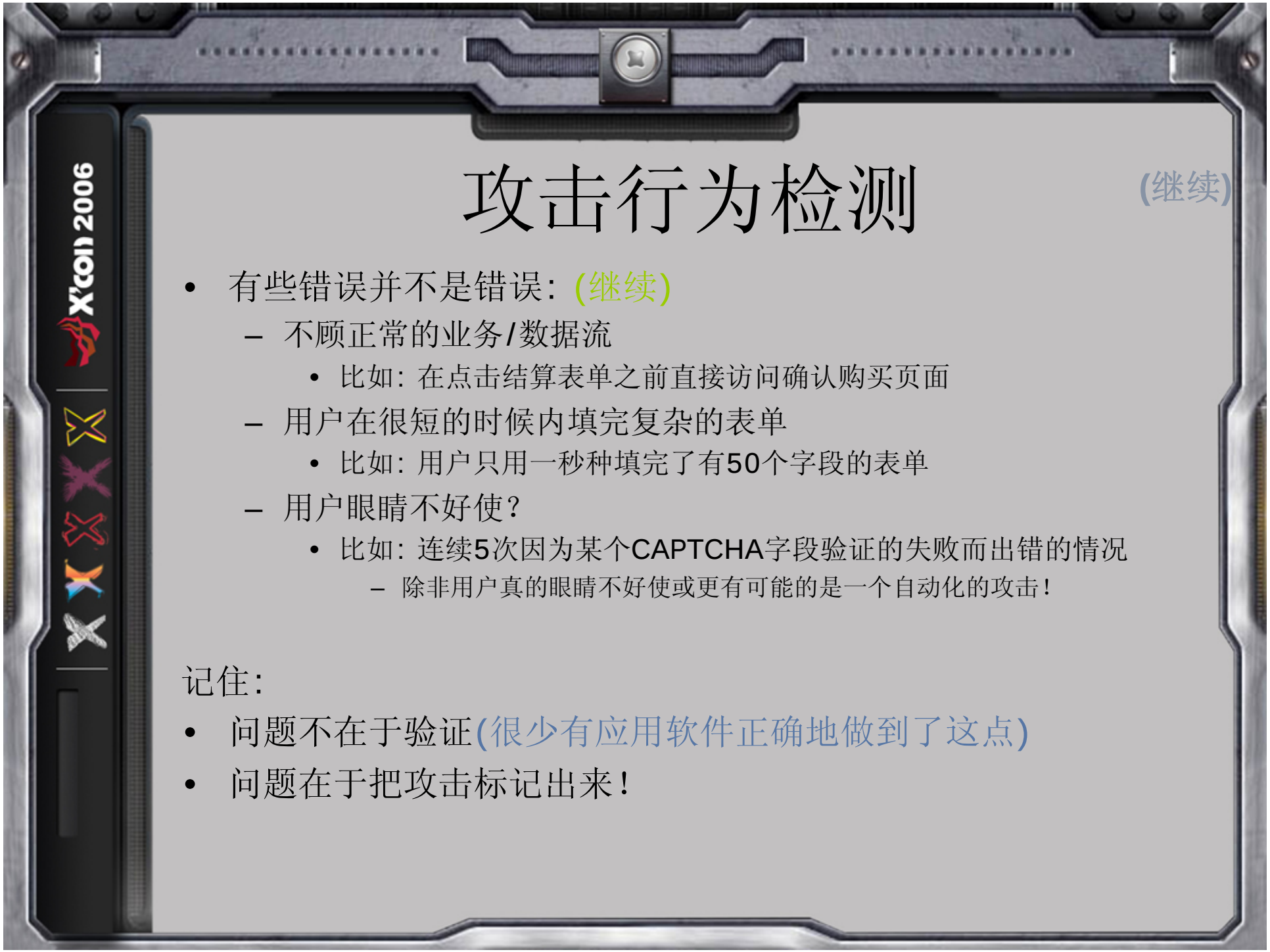


攻击行为的检测

- 正常用户不会重复地犯错误
  - 一些重复的错误明显就是攻击
  - 应该触发告警或其他的动作

- 有些错误并不是错误：
  - 认证错误：
    - 比如：1分钟内超过5次的用户名/口令认证失败
  - 验证错误：
    - 比如：明显的SQL语句串而不是电子邮件地址
    - 比如：明显的SQL语句串而不是数值类型的itemID
    - 比如：HTML/JavaScript的保留字而不是一个姓名
    - 比如：明显的缓冲区溢出攻击(比如:字串超过200字节长)

用户错误？ 当然不是！很肯定的说就是攻击。



攻击行为检测 (继续)

- 有些错误并不是错误: (继续)
  - 不顾正常的业务/数据流
    - 比如: 在点击结算表单之前直接访问确认购买页面
  - 用户在很短的时候内填完复杂的表单
    - 比如: 用户只用一秒种填完了有50个字段的表单
  - 用户眼睛不好使?
    - 比如: 连续5次因为某个CAPTCHA字段验证的失败而出错的情况
      - 除非用户真的眼睛不好使或更有可能的是一个自动化的攻击!

记住:

- 问题不在于验证(很少有应用软件正确地做到了这点)
- 问题在于把攻击标记出来!

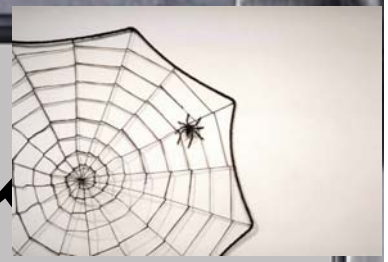
HAIDS → HAIPS

- 攻击检测的几个目标(并不只是默默地阻止它):
  - 知道你的应用正在遭受攻击
    - 你没意识到....!!!
  - 知道谁在执行攻击
  - 知道攻击者在试图得到什么
    - 所以你能知道什么看起来比较脆弱需要更多关注
  - 阻断当前或将来的攻击
    - 自动锁定帐号
    - 逮捕或起诉攻击者
    - 其他可能的动作
- 把一个基于主机的应用层入侵检测系统转变为一个阻断系统，我们需要采取行动而不仅仅是告警...

HAIDS → HAIPS

- 一个HAIPS需要实现的动作:
  - 记录攻击的细节
    - 发送一个一般性的不泄露信息的错误消息给客户端
    - 在服务端记录一个完整准确的错误信息
  - 给应用软件管理员发送包含完整错误信息的邮件
  - 给安全部门发送包含完整错误信息和会话的邮件
  - 发送短信
  - 锁定用户帐号
  - 发出一个挑战操作来阻止和验证自动化的攻击
    - 可以是一个CAPTCHA 或其他必须由人来识别处理的问题
  - 重定向到一个告警页面
    - *什么时候警告攻击者?*
  - *更有趣的...*
    - 重定向到一个蜜罐
    - 在接下来的5分钟向此用户发送垃圾数据

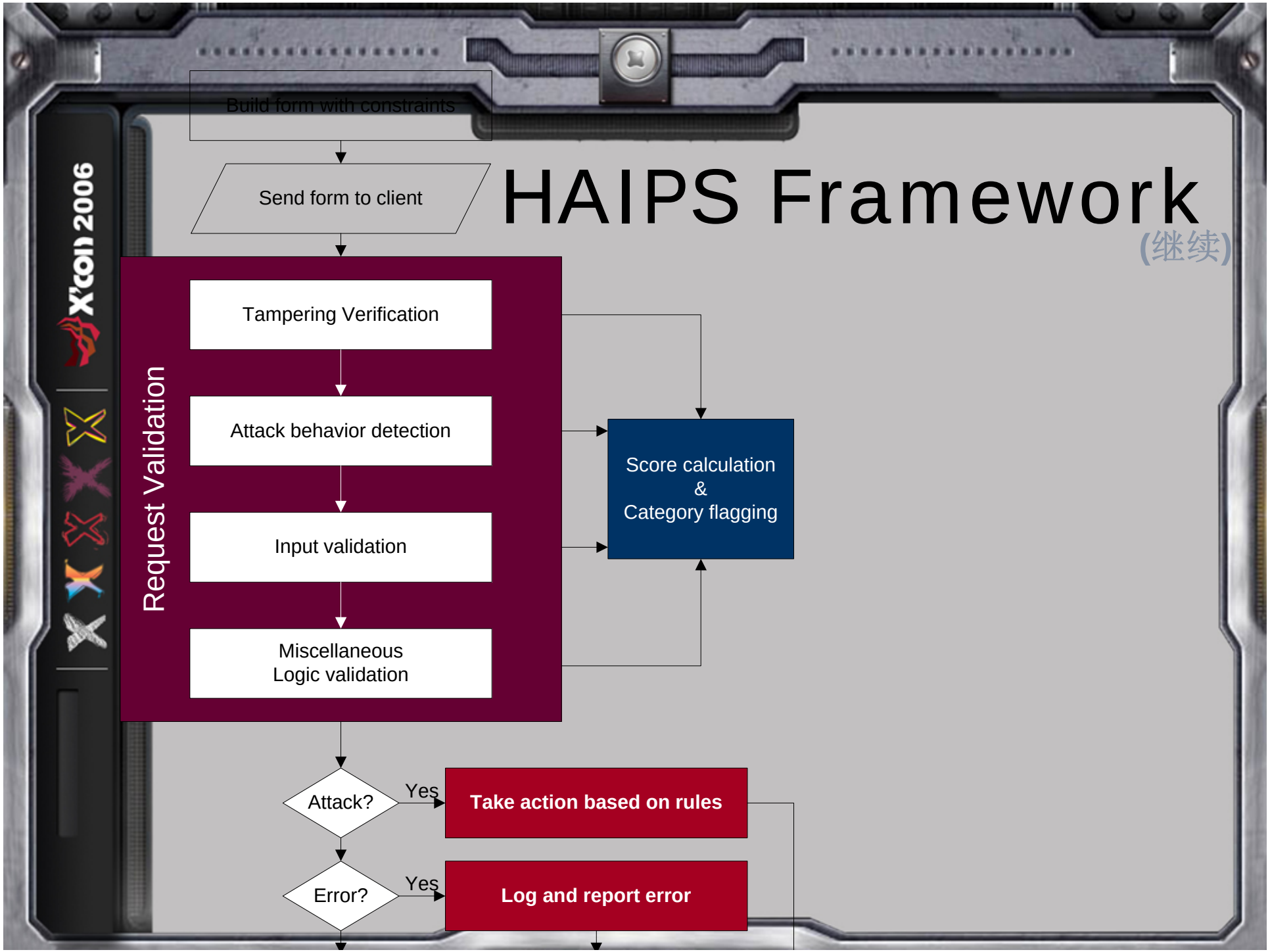
HAIPS Framework



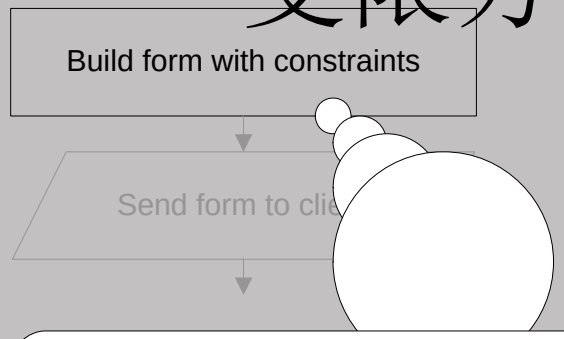
- HAIPS必须在每个你要保护的应用中植入
- 最好以framework的形式实现

- 我们试着提出这样的一个framework

HAIPS Framework (继续)



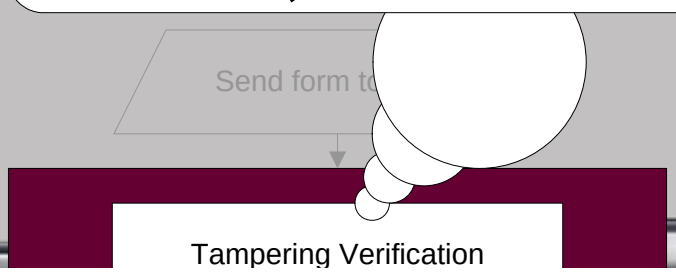
HAIPS Framework
受限方式创建表单



- 标记隐藏的字段
- 标记相关字段的最大长度
- 自动生成客户端JavaScript验证代码
- 记住cookies, 隐藏字段等不应该被修改的值
  - 将来我们可以验证它们...

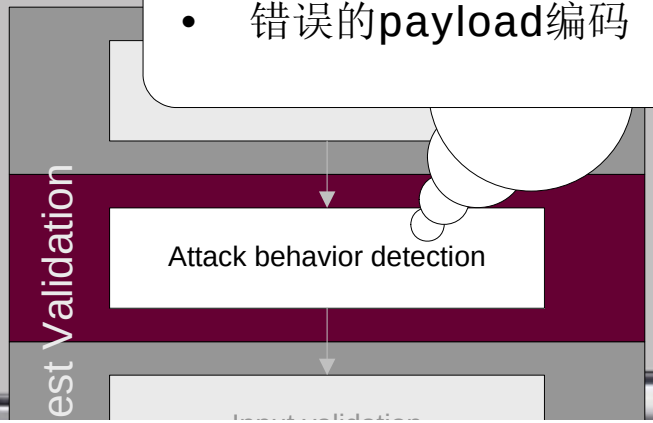
HAIPS Framework
恶意修改的检查

- 检查不可修改字段有没有被修改:
  - 下拉列表
  - 按钮
  - 复选框
  - 隐藏字段
  - 固定长度的文本输入框
  - cookies
- 检查某些必须存在的字段
  - 如果没有的话，肯定是被故意删掉了…



HAIPS Framework
攻击行为检测

- 连续的错误
- SQL注入
- 跨站脚本
- 缓冲区溢出
- cookie缺失
- 无效或缺失的referrer
- 会话中user-agent的改变
- 缺失的参数
- 错误的请求方法
- 错误的payload编码
  - 错误的头部编码
  - 可疑的URL
  - 触发的陷井
  - 其他典型的注入攻击
  - 不期望有的参数
  - 角色绕过企图
  - 其他绕过客户端验证的企图



HAIPS Framework (继续)

攻击行为检测

- 连续的错误
  - 比如: 1分钟内5次登录失败
- SQL注入
  - 比如: 数据中包含‘或其他SQL关键字
- 跨站脚本
  - 比如: 名字内包含 <script> 或其他典型的 XSS 特征
- 缓冲区溢出
  - 比如: 参数串长度超过预期的两倍
- 缺失的cookies
  - 比如: 表单中的themeID cookie缺失
- 缺失的referrer
  - 比如: 攻击者使用了会过滤此字段的代理或设置浏览器忽略它们。惩罚攻击者!

HAIPS Framework (继续)

攻击行为检测

- 缺失的参数
  - 比如：某个强制使用的参数缺失，JavaScript阻止了用户提交表单
- 在会话中修改user-agent
  - 比如：用户用Firefox登录到应用，而在这之后却声称自己是IE...
- 无效的动作
  - 比如：请求预期是POST的，但却是以GET提交的
- 错误的payload编码
  - 比如：表单预期是以application/x-www-form-urlencoded方式编码实际却是multipart/form-data

HAIPS Framework
攻击行为检测

(继续)

- 错误的头部编码
  - 比如: 攻击没有在请求中正确地进行URL编码...
- 可疑的URL
  - 比如: URL包含的参数以 / 开始或包含 ../
  - 比如: URL包含保留的文件名
    - web.config
    - WEB-INF
    - .bak
    - blahblahblah~ (Unix风格的备份文件)

HAIPS Framework
攻击行为检测

(继续)

- 设置的陷井被触发
  - “must .... press .... red .... button.... !!” or
  - “I wonder what this is for?”
  - 比如:

```
<form action="login.jsp" method="post">
<input name="username" maxlength="10" />
<input name="password" type="password"
       maxlength="100" />
<input type="submit" value="Login" />
<!-- <input type="hidden"
            name="is_admin"
            value="0" /> -->
</form>
```

- 攻击者可能会受引诱
  - 去掉上面的注释设置 **is_admin** 为 1
    - 去掉注释明显就是触发我们陷井...

HAIPS Framework
攻击行为检测

(继续)

- 设置的陷井被触发
  - 更多的例子:
    - 用户试图以‘admin’登录 → ‘admin’
      - … 而你知道admin并不是那么叫的
    - 已认证的普通用户试图访问**/admin**
      - … 而你知道管理目标是在 **/a**
      - … 而那家伙并不是一个admin!
- 有谁在评测安全的过程中从没有用以上的方法碰碰运气?

HAIPS Framework
攻击行为检测

(继续)

- 其他典型的注入攻击
  - DAP/LDAP注入 比如: 一个名字包含*或，
  - CR/LF 注入 比如: 一个名字包含 CR 或 LF.
  - Shell命令注入 比如: 一个用户名包含 ；
  - XPath 注入 比如: 一个用户名包含‘ 或 =
  - Cobol字段注入... 开玩笑呢 :-)

- 没有预期要存在的多余参数
  - 有时候攻击者会通过在请求中加入想像中可能会存在参数碰运气：
    - 比如: `is_admin=1`
    - 比如: `loggedon=true`
    - ...

HAIPS Framework
攻击行为检测

(继续)

- 角色绕过企图
  - 比如：一个以普通帐号登录的用户试图直接访问在他的面板中不可能出现的管理接口URL
- 其他客户端验证的绕过
  - 如果用户绕过无论如何多细小的JavaScript验证例程，则意味着他在攻击！
  - 之前我们已经讨论了其中的大多数情况
- 尽情加入你们自己用于区分手工构造的攻击操作和用户在浏览器中的点击...

HAIPS Framework

输入验证

验证所有的数据类型

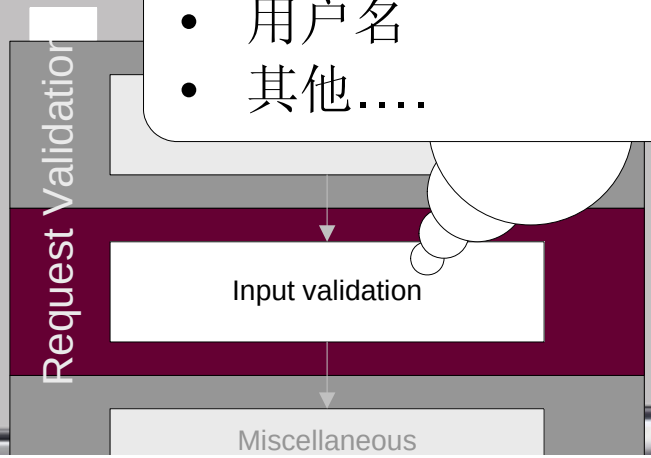
- 日期
- 邮政编码
- 电话号码
- 地址
- 名字
- 总数
- 电子邮件地址
- 用户名
- 其他....

某些情况可以识别为被入侵:

- 从JavaScript日历中选取的日期不会有错
- 如果错了就意味着攻击...

有些情况不能:

- 比如: 没有办法知道诸如 *thisisabadname* 这样的名字输入是否为一个真实的名字



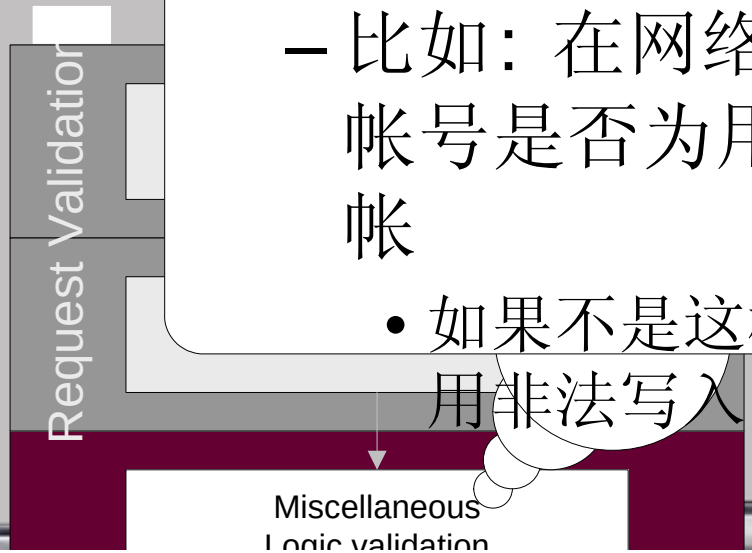
HAIPS Framework
攻击行为检测

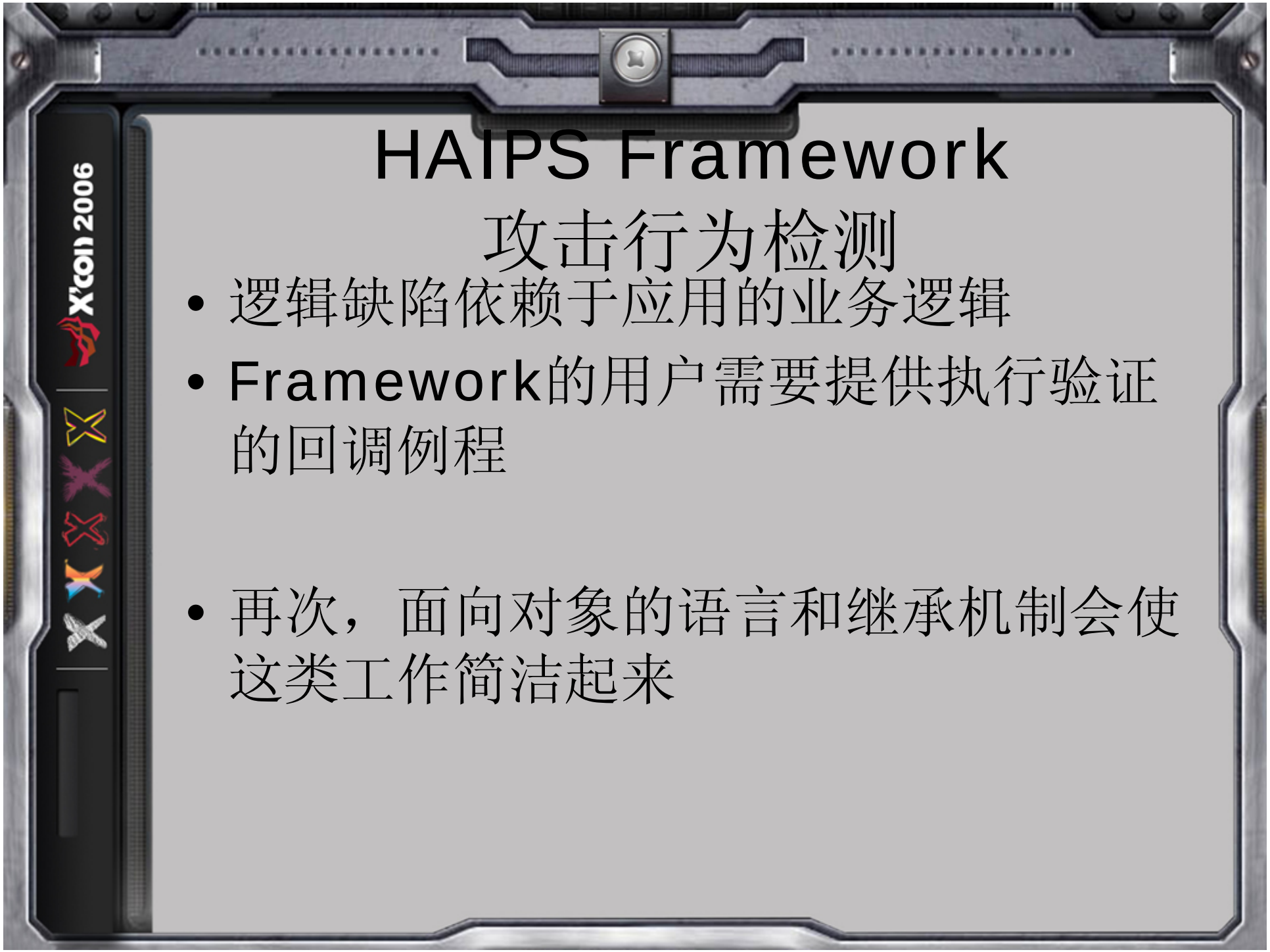
- 需要验证很多不同类型的数据
  - Framework的用户必须为每个类型检查提供回调例程
  - Framework的创建者可以预定义一些典型的类型
  - Framework的用户只需要添加framework中还未提供的类型
  - 面向对象的语言和继承机制使这类工作简洁起来

HAIPS Framework

多样的逻辑验证

- 检测逻辑攻击取决于你
  - 比如：在网络银行中，一般是先验证帐号是否为用户所有然后才发送细节
    - 如果不是这样，则意味着用户在试图利用非法读取的逻辑缺陷
  - 比如：在网络银行中，一般是先验证帐号是否为用户所有然后才会执行转帐
    - 如果不是这样，则意味着用户在试图利用非法写入的逻辑缺陷

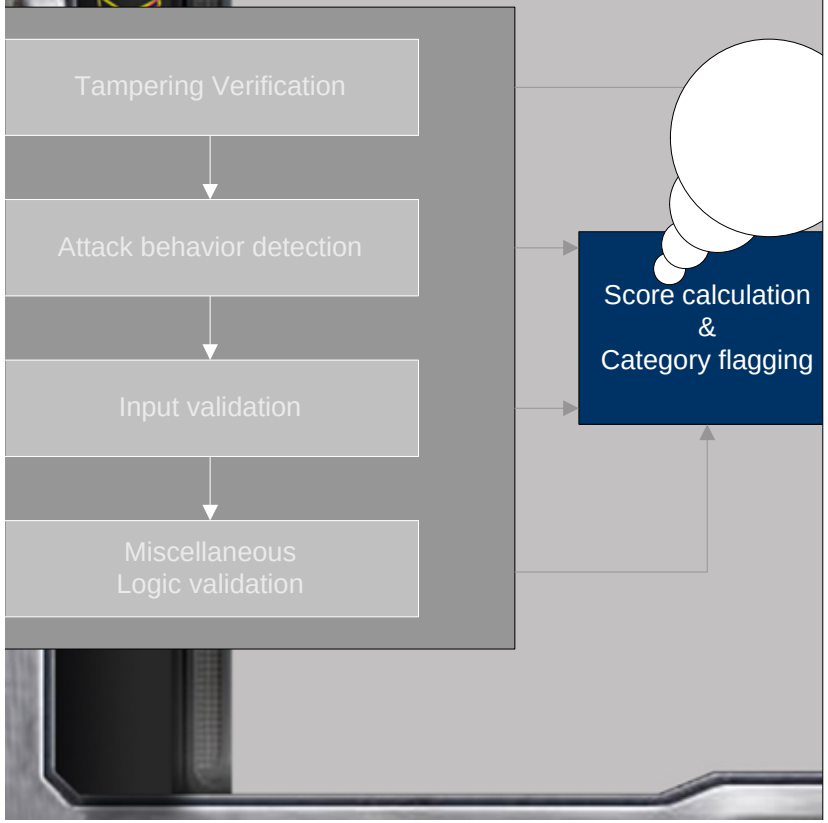




HAIPS Framework
攻击行为检测

- 逻辑缺陷依赖于应用的业务逻辑
- Framework的用户需要提供执行验证的回调例程
- 再次，面向对象的语言和继承机制会使这类工作简洁起来

HAIPS Framework
打分 & 分类

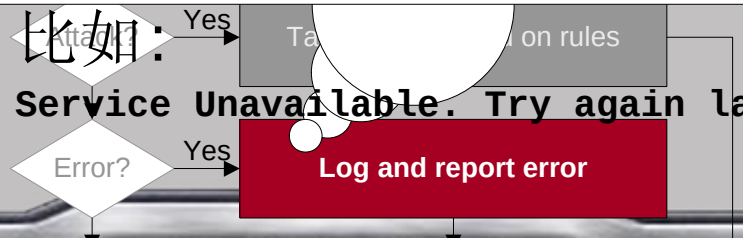


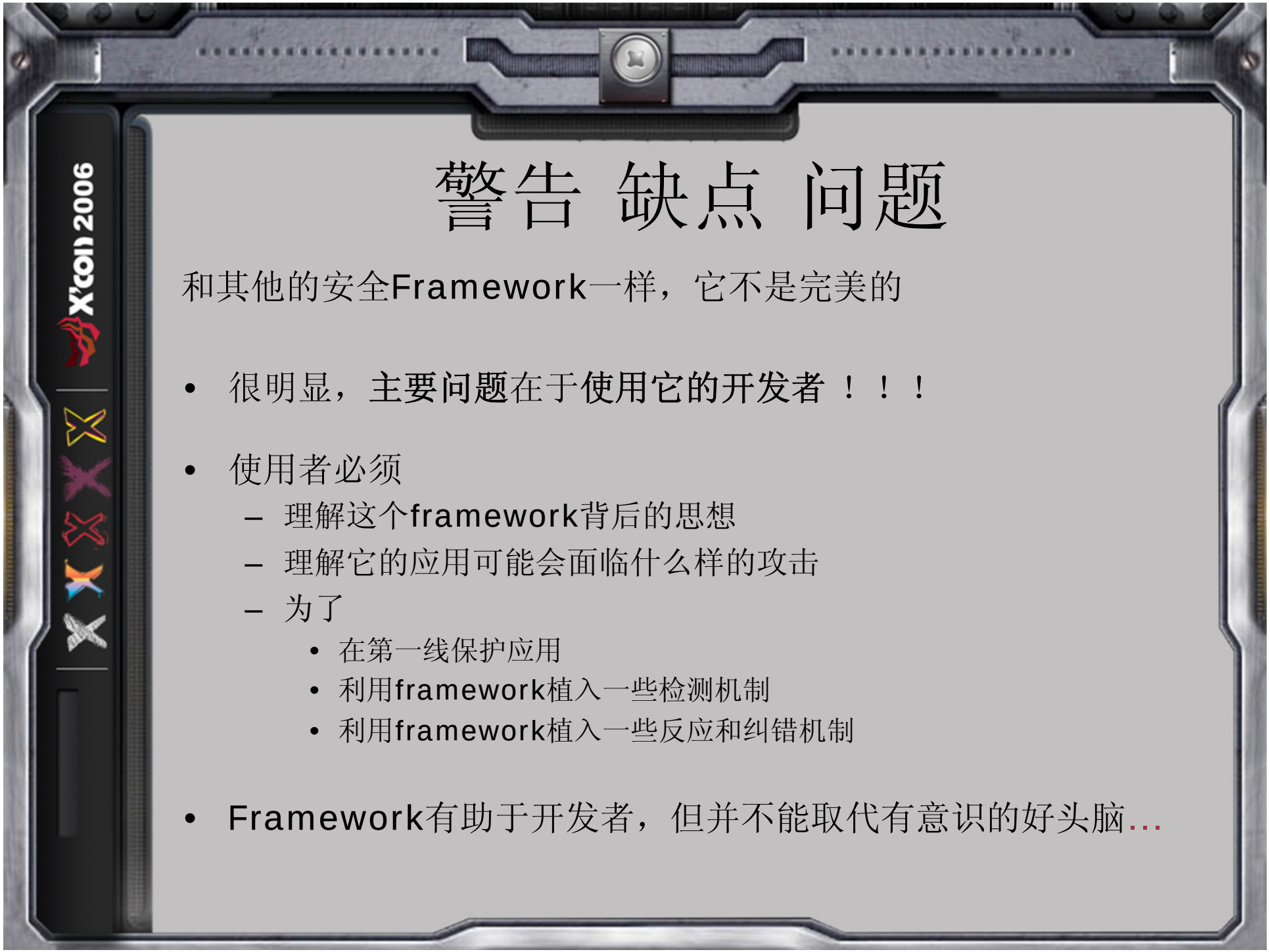
- 打分:
  - 请求中每个上面提及的消极因素累加到一个攻击评分
  - 越恶意的攻击评分越高
  - 允许应用的所有者设置告警和阻断的阀值
  - 通过为已知的浏览器问题设置负分来避免误报
- 分类标记:
  - 简单的说，就是确定“正常”情况下的错误还是一次攻击

HAIPS Framework

基于规则采取回应动作

- 记录和报告错误是非常简单明了的
  - 记录一条非常详细的错误消息到文件或数据库，包含所有可能的信息:
    - 比如:
    - 时间/日期
    - 用户名，IP地址
    - 原由：比如：**SQL注入 username=x' or 1='1**
  - 发送一个一般性的错误信息给客户端
    - 比如：
    - **Service Unavailable. Try again later.**





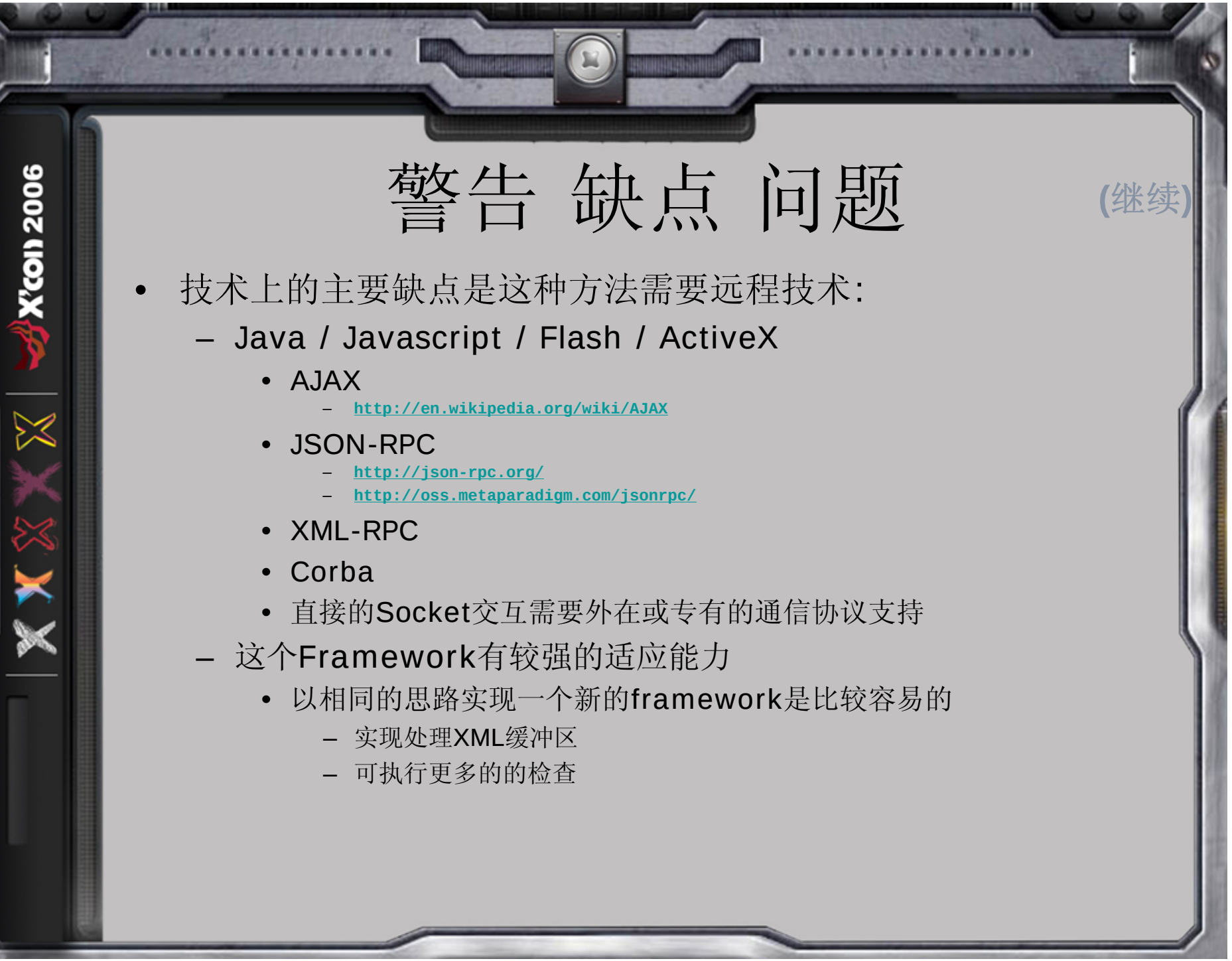
警告 缺点 问题

和其他的安全Framework一样，它不是完美的

- 很明显，**主要问题**在于**使用它的开发者 ！！！**
- 使用者必须
  - 理解这个framework背后的思想
  - 理解它的应用可能会面临什么样的攻击
  - 为了
    - 在第一线保护应用
    - 利用framework植入一些检测机制
    - 利用framework植入一些反应和纠错机制
- Framework有助于开发者，但并不能取代有意识的好头脑...

警告 缺点 问题 (继续)

- Framework只能保护将其植入的应用
  - 它不能魔术般的保护现有的应用
- Framework只能保护编写质量尚可的应用
  - 一些应用本身就违背自己的规则，所以framework会错误地把不寻常的活动标记为攻击
- Flash格式支持情况如何?
  - 它们可以被支持
    - 当然需要告诉Framework额外的信息
      - 关于表单的内容
      - 不可变字段值
      - 各种其他参数，比如内容编码和请求方法

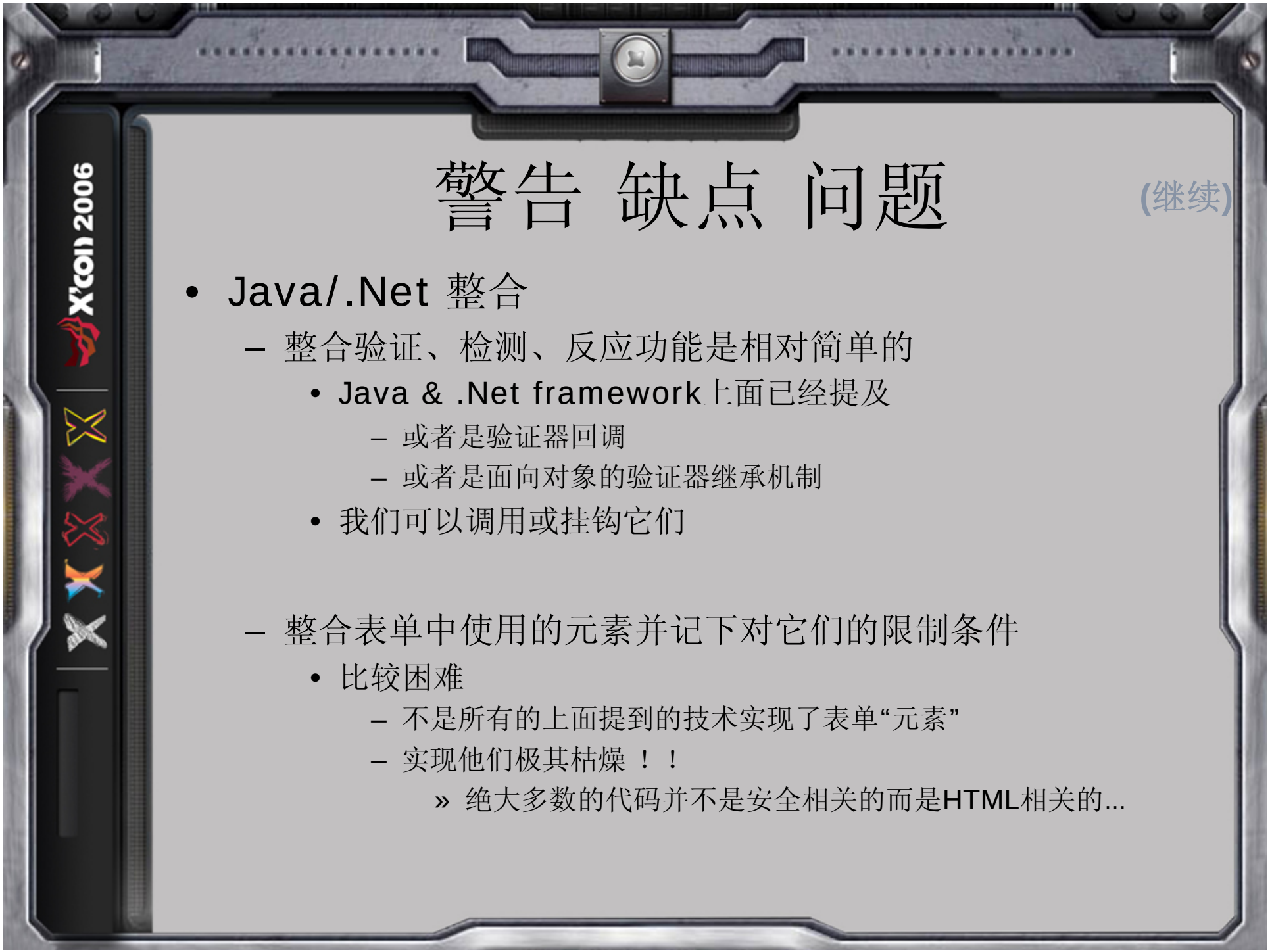


警告 缺点 问题 (继续)

- 技术上的主要缺点是这种方法需要远程技术:
  - Java / Javascript / Flash / ActiveX
    - AJAX
      - http://en.wikipedia.org/wiki/AJAX
    - JSON-RPC
      - http://json-rpc.org/
      - http://oss.metaparadigm.com/jsonrpc/
    - XML-RPC
    - Corba
    - 直接的Socket交互需要外在或专有的通信协议支持
  - 这个Framework有较强的适应能力
    - 以相同的思路实现一个新的framework是比较容易的
      - 实现处理XML缓冲区
      - 可执行更多的的检查

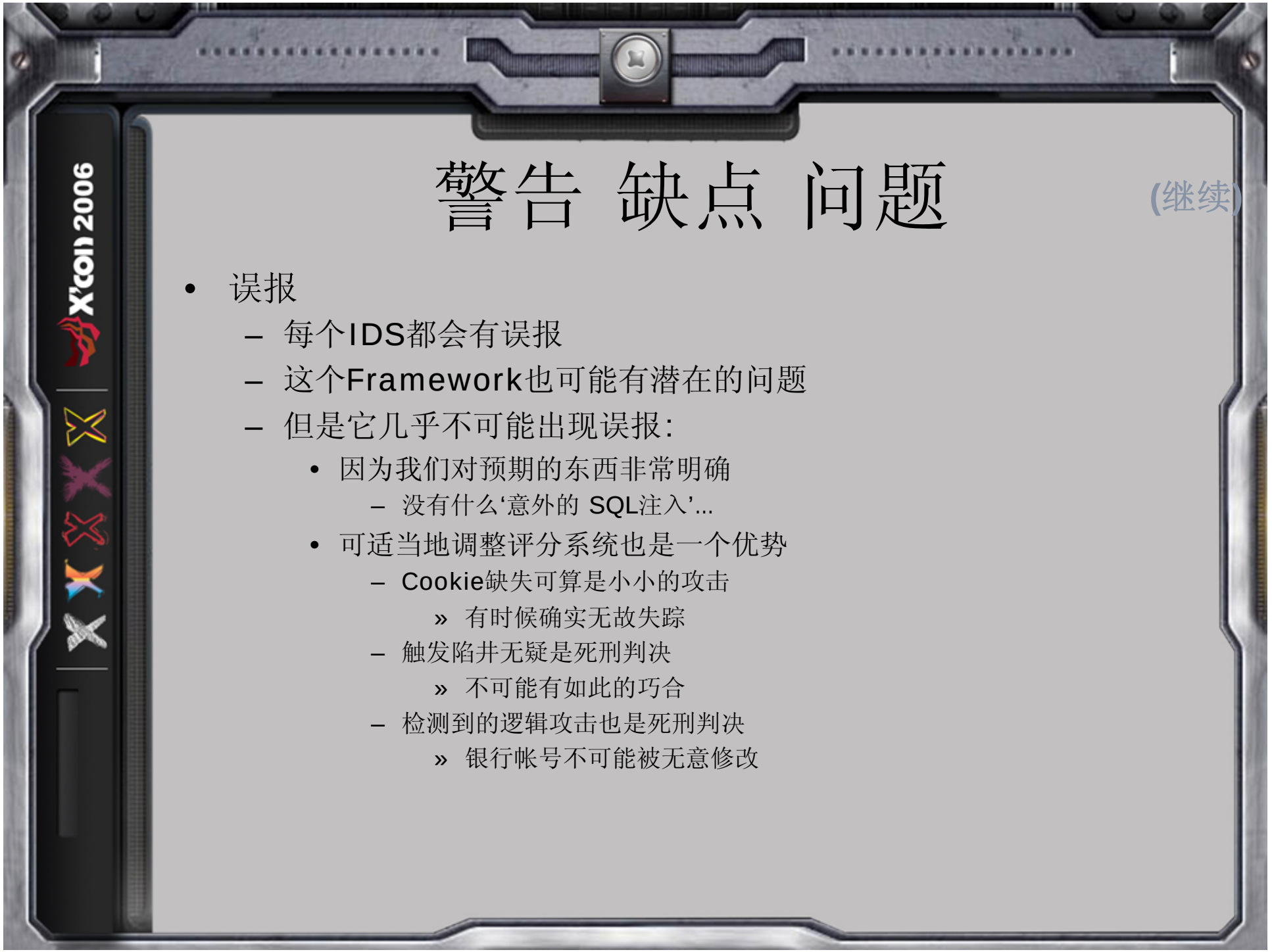
警告 缺点 问题 (继续)

- 这个Framework必须与现有Framework整合起来
- Java:
  - Struts, Java Server Faces, Tapestry, OWASP Stinger,其他...
    - 我们的Framework需要集成他们所有呢还是只处理其中的一个呢?
    - 用Framework集成HIPS还是反过来?
- .Net:
  - 内置的 .Net 验证机制
- 每个应用平台需要单独整合..



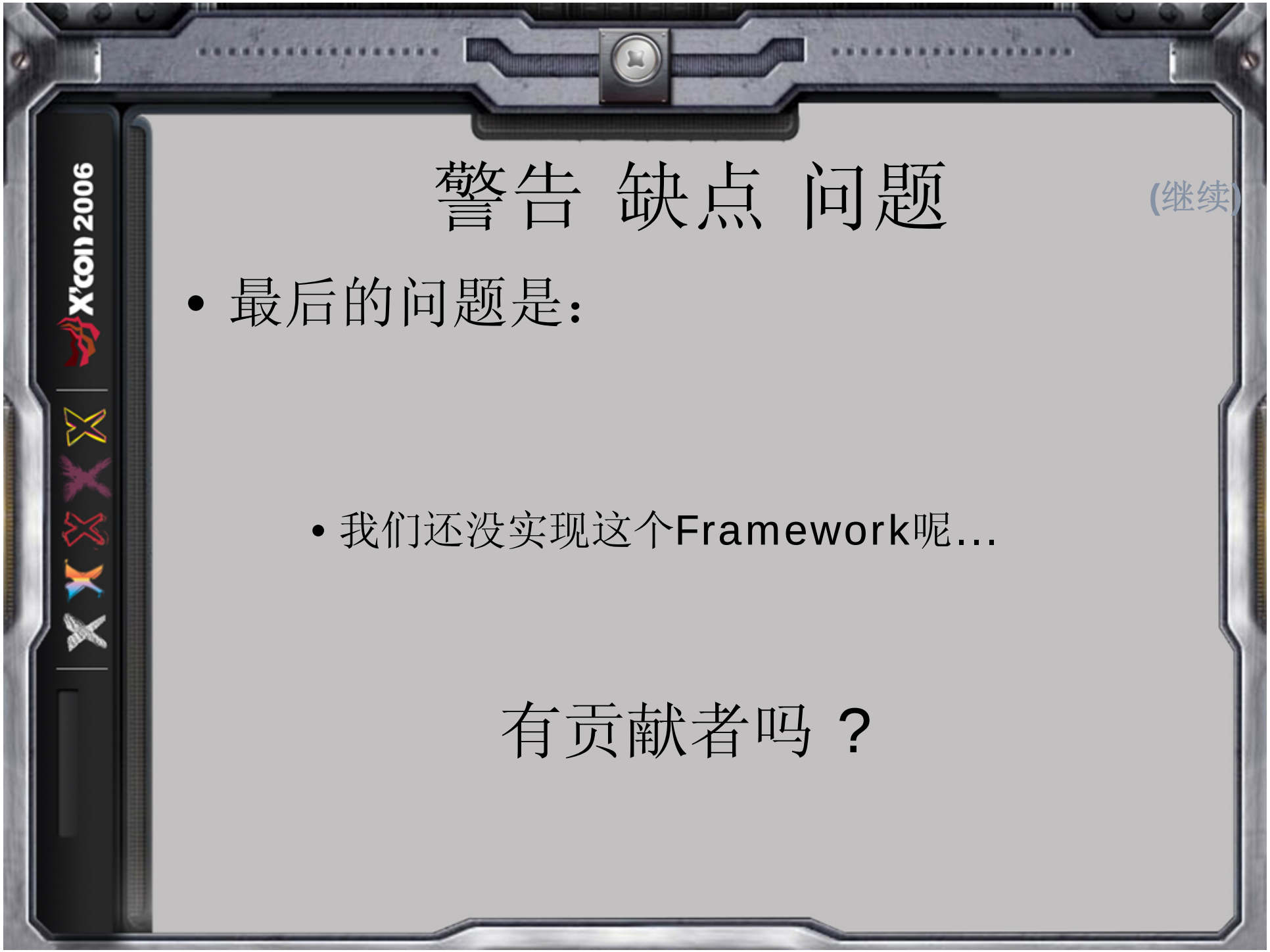
警告 缺点 问题 (继续)

- Java/.Net 整合
  - 整合验证、检测、反应功能是相对简单的
    - Java & .Net framework上面已经提及
      - 或者是验证器回调
      - 或者是面向对象的验证器继承机制
    - 我们可以调用或挂钩它们
  - 整合表单中使用的元素并记下对它们的限制条件
    - 比较困难
      - 不是所有的上面提到的技术实现了表单“元素”
      - 实现他们极其枯燥 ！！
        - » 绝大多数的代码并不是安全相关的而是HTML相关的...



警告 缺点 问题 (继续)

- 误报
  - 每个IDS都会有误报
  - 这个Framework也可能有潜在的问题
  - 但是它几乎不可能出现误报:
    - 因为我们对预期的东西非常明确
      - 没有什么‘意外的 SQL注入’...
    - 可适当地调整评分系统也是一个优势
      - Cookie缺失可算是小小的攻击
        - » 有时候确实无故失踪
      - 触发陷井无疑是死刑判决
        - » 不可能有如此的巧合
      - 检测到的逻辑攻击也是死刑判决
        - » 银行帐号不可能被无意修改

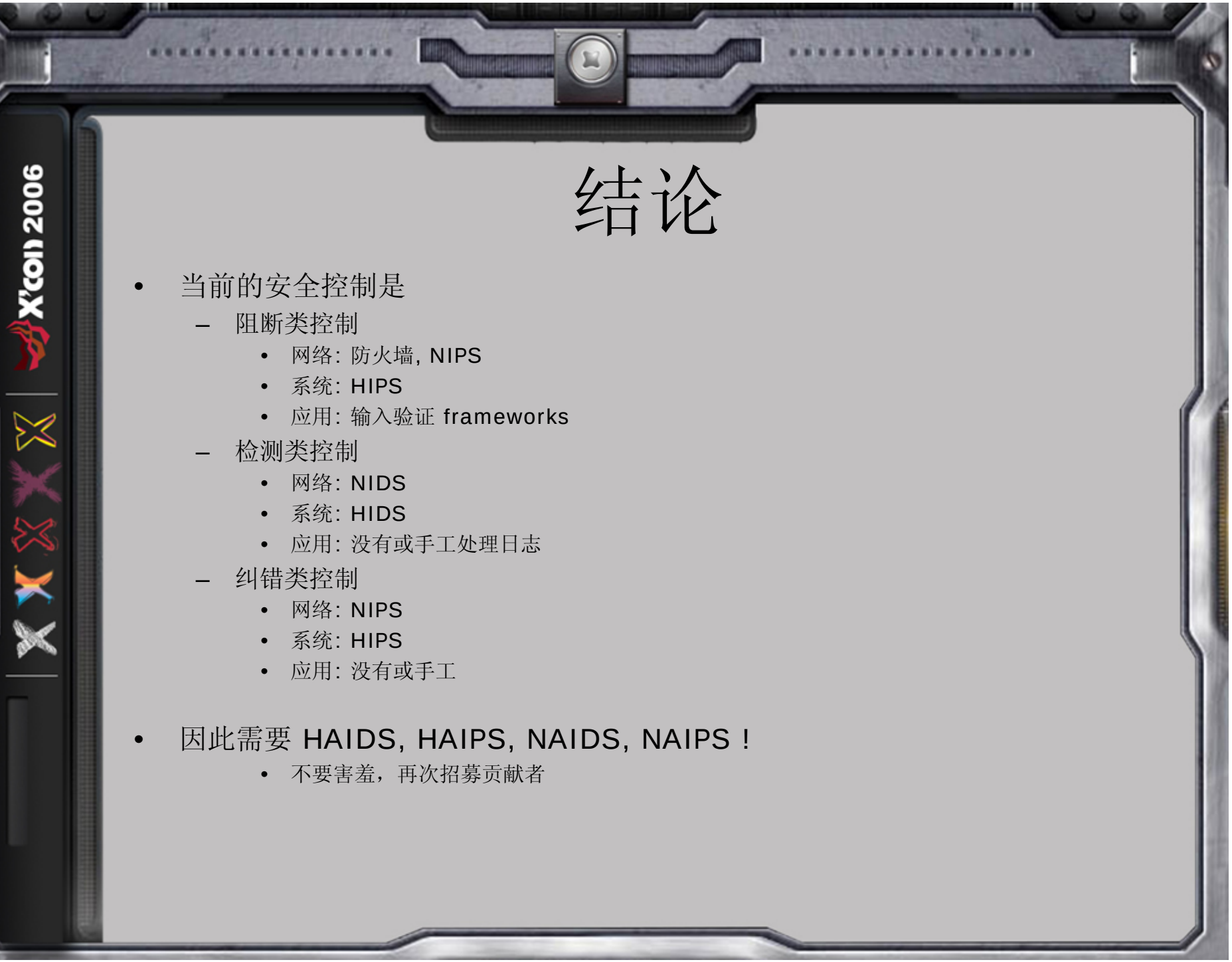


警告 缺点 问题

(继续)

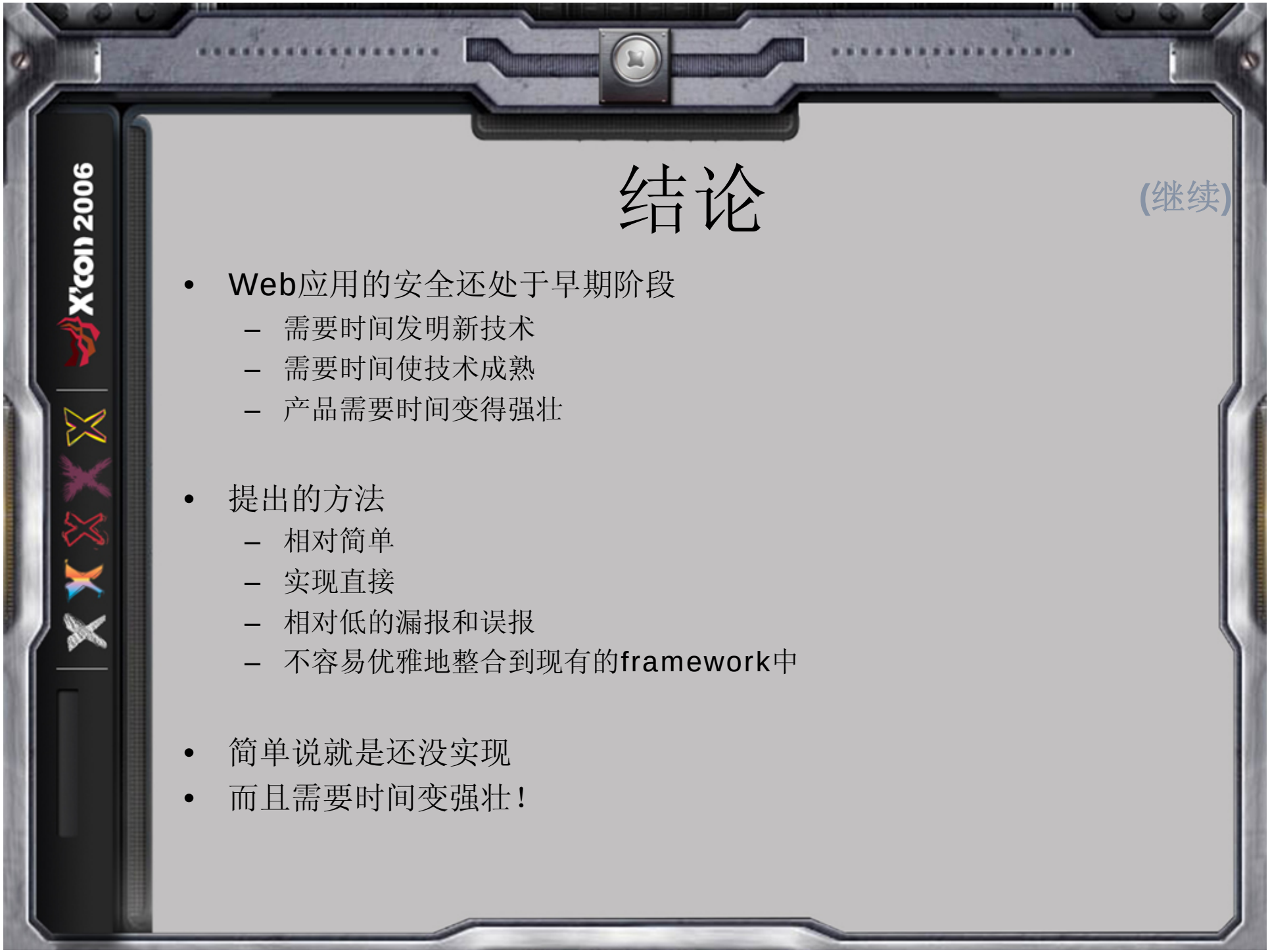
- 最后的问题是:
  - 我们还没实现这个Framework呢...

有贡献者吗？



结论

- 当前的安全控制是
  - 阻断类控制
    - 网络: 防火墙, NIPS
    - 系统: HIPS
    - 应用: 输入验证 frameworks
  - 检测类控制
    - 网络: NIDS
    - 系统: HIDS
    - 应用: 没有或手工处理日志
  - 纠错类控制
    - 网络: NIPS
    - 系统: HIPS
    - 应用: 没有或手工
- 因此需要 HAIDS, HAIPS, NAIDS, NAIPS !
    - 不要害羞，再次招募贡献者



结论 (继续)

- Web应用的安全还处于早期阶段
  - 需要时间发明新技术
  - 需要时间使技术成熟
  - 产品需要时间变得强壮
- 提出的方法
  - 相对简单
  - 实现直接
  - 相对低的漏报和误报
  - 不容易优雅地整合到现有的framework中
- 简单说就是还没实现
- 而且需要时间变强壮！

提问 ??