针对恶意混淆脚本的模拟执行检测研究

乔 伟[1] 潘宣辰[1] 肖新光[2] 李柏松[2]
(武汉安天信息技术有限责任公司 武汉 430070)[1] (北京安天电子设备有限公司 北京 100084)[2]

摘 要 随着Web的高速发展,一大批新兴应用层出不穷。而这些应用的背后,依赖于与脚本和浏览器插件技术的发展。与此同时,恶意代码编写者开发出大量恶意脚本代码,并通过多种混淆手段对恶意脚本进行混淆和变形,逃避以特征码检测技术为主代表的恶意代码检测,其中尤其以Javascript混淆代码为巨。通过各种混淆方式,可以产生大量恶意代码的变种,并广泛撒网式的传播方式威胁网民的安全。这种方法也大大干扰了恶意代码的检测,成为整个Web恶意代码中最为艰巨的防御点。针对这一情况,从脚本本身的特点出发,使用模拟执行、自动检测的方法,可以有效地对混淆代码进行反混淆并检测,利用这种方式可以达到较好的检测效果。

关键词 混淆脚本,恶意脚本检测,模拟执行
中图法分类号 TP393 **文献标识码** A

Virtual Execution-based Detection Research on Obfuscated Scripts

QIAO Wei[1] PAN Xuan-chen[1] XIAO Xin-guang[2] LI Bai-song[2]
(Wuhan Antiy Information Technology CO., LTD., Wuhan 430070, China)[1]
(Beijing Antiy Electronic Equipment CO., LTD., Beijing 100084, China)[2]

Abstract With the rapid development of Web technology, more and more Web application come true. These applications depend on the scripts and plugins. At the same time, malicious code writers make a large number of malicious script code, and obfuscated them to evade the detection by signature-based detection. In these malicious scripts, Javascript scripts make a large quantity. With the obfuscation, code writer can make large number of variants, and transmission on the Web which has become the great threat for Web security. In response, we researched on the basic Feature of the scripts and virtual executed them to decrpt and detect them. With these technology, obfuscated scripts detection made a great improvement.

Keywords Virtual execution, Malicious scripts detection, Obfuscated scripts

1 概述

随着Web攻防的快速发展,防御者面对的一个主要问题是,脚本形态的恶意代码是以源代码形式存在,具有二进制代码所不具有的丰富语义。在不改变恶意代码功能的情况下,代码编写者只需要进行一些代码调整便可以产生大量变种。这种脚本常常被称为混淆脚本。本文主要针对Javascript混淆脚本,通过对混淆代码的大量详实而细致的分析工作,研究存在的一些共性特征及特殊情况。在此基础上,通过模拟执行等手段,混淆代码被还原为没有经过变换的代码,并能够基于静态特征码的方法进行检测。通过这种方法,本文完成了一个针对恶意混淆代码的自动检测系统,并对其效果进行了一些实验和分析。

本文第2节讲述本文的研究背景及相关研究状况;第3节对恶意代码即我们的研究对象进行了详实的分析;第4节针对混淆代码提出本文的检测方法及实现方案;第5节对系统的实验和分析;最后总结全文。

2 研究状况

2.1 Javascript混淆技术研究

近年来由于Javascript恶意代码大量使用混淆技术,为了对此进行检测和防御,有很多人开始对Javascript混淆技术进行研究。

Jose Nazario在CanSecWest 2007上介绍了恶意代码使用Javascript混淆技术的原因,以及FromCode,Base64 encoding,String splite,Customer encoder等混淆方式,并介绍了如何使用NJS和Spidermonkey[1]等工具对混淆代码进行分析[2]。

本文受863国家基金项目(2009AA01Z436)资助。

乔 伟(1986—),男,硕士,主要研究方向为网络安全等,E-mail:dhqway@antiy.com;潘宣辰(1987—),男,硕士,主要研究方向为网络安全等;肖新光(1974—),男,硕士,主要研究方向为网络安全等;李柏松(1975—),男,硕士,主要研究方向为网络安全等。

Wayne Huang 在 OWASP 2008 上的报告提到 Name Obfuscation, String Splite, Code Encrytion 等变形技术，也介绍了环境检测、反调试、枚举函数接口、代码自校验等反检测和反分析技术[3]。

Kolisar 在 Defcon 16 上的报告主要介绍了常见的加密方式，并指出这些方式都会用到一些关键函数 eval(), unescape() 等，在此基础上他提出使用对对象的成员枚举(Member Enumeration)的方式来获取这些函数接口，从而增加检测难度[4]。

2.2 针对恶意脚本的检测方法

由于恶意代码会对系统或者浏览器环境进行破坏活动，因此通过对这些可能的危险行为的监控可以检测恶意代码。

Symantec 公司的反病毒软件 Norton AntiVirus 中采用了 Script Blocking[5] 技术，即通过检测脚本修改注册表的行为来检测恶意脚本。国外的 Frank Apap 等人、国内的鲍欣龙等人也做过类似研究[6,7]。

Caffeine Monkey[8] 这款 Javascript 检测引擎与本文的思想有相似之处。它们的引擎基于 Mozilla 的 Spidermonkey 之上。通过对原有 Javascript 引擎的关键函数挂钩的方式，可以实现对加密变形脚本的解密，并在运行时中进行日志记录。它对于脚本的检测考虑得不够全面。

ADSafe[9] 不允许 eval 等动态执行方式，并使用了一些静态检查技术。但也由于这一点，导致了很多正常的 Javascript 功能无法使用。FBJS[10] 是 Facebook 公司使用的一项技术，通过对 Javascript 语言的一些特性进行限制，比如对 DOM 访问等来进行检测和防御。使用 FBJS 已经成功地阻止了一些常见攻击[11]。

除了在静态语言级别进行限制，还可以在运行时中进行限制。Caja[12] 试图通过运行时检查的方式限制 Javascript 的功能。BrowserShield[13] 也是使用这种技术对浏览器中的脚本进行重写，确保 Javascript 和 HTML 都是安全的。

Christopher Anderson[14] 等人建立了 JS0 语法的类型系统，通过定义变量类型、约束和求解、约束生成等一系列手段来检查 Javascript 代码。Dachuan Yu[15] 等人在此基础上使用基于程序插庄(program Instrumentation)来检测 Javascript 的恶意行为。

3 检测对象研究

为了逃避检测，恶意代码使用多种方式逃避检测。主要的方式有编码、加密、混淆、变形、切分等。

3.1 编码

Javascript 本身提供了 unescape, escape, fromcharcode 等编解码函数，利用这些函数可以将类似“%3c%73%63%72%69%70%74%3E”解码为“<script>”，另一方面，eval, document. write 等函数还能自动将传入参数进行解编码，例如 document. write(“\x3c\x73\x63\x72\x69\x70\x74\x3E”)与 document. write(“<script>”)等同。

例如 Trojan-Downloader. JS. Agent. ceu 就利用了这样一种方式。它的变形形式为：

```
Var Words = “编码”
document. write(unescape(Words))
```

3.2 加密

病毒编写者不满足于使用简单的编码方法，进而引入加密方法，使用 xor, replace 等字串变换函数编写一些加解密函数，例如样本 Trojan-Downloader. JS. Agent. bra 甚至用到了 md5 算法。

除了利用自定义的加密函数，也可利用系统提供的加密方法对脚本进行加密。通过将 script 便签中的 language 字段指定为 JScript. Encode，即可以使用 JSEncode 加密方法。浏览器会对这种加密方法的样本进行解密，并执行其中的内容。这种加密方法的典型特点是整个代码段以“#@～^”4 个字母开头，而以“^#～@”4 个字母作为结束。不过这种加密方法依赖于客户端浏览器，只有 Internet Explorer 支持。

3.3 混淆

插入正常代码，即将恶意代码拆分为两段夹杂到正常代码中间，恶意代码中放置注释、无效代码等。可以利用浏览器注释方式“<! —”、“--〉”和脚本注释方式“//”夹杂在样本中作为妨碍分析的手段。

恶意代码也可以在中间插入二进制 0 或者除去换行符等格式字符，因为这样的字符浏览器和编译器处理时会忽略。

有时甚至插入一些语法正确，但无实际意义的变量，因为脚本引擎的健壮性和容错性可以保证这样的样本仍然被脚本引擎正常处理。例如 vO86 = " IH\&HfMB2"; 6. 247115E-02, kT27 = ". 7464055"这段脚本中的 6. 247115E-02 是毫无意义的，能够被脚本引擎正常处理。

这样一些方式无法真正逃过静态检测和病毒分析人员的法眼，但是会严重干扰分析人员的工作，同时会影响特征码的有效提取。利用这样一些简单的手段，就可以很大程度上挫败使用单串检测和固定位置特征码的反病毒引擎。

3.4 变形

对加密的更高形态是利用变量替换、字串拼接、字串替代、移位等各种方式，最终的效果如同乱码一般。这样的样本不便于人工分析，也就不易提取特征码。针对这样的代码需要去伪存真。

变量替换是只将字串、变量甚至函数、对象等赋值到一个新的变量，然后使用这个新的变量进行后面的操作。如下面的示例中，将恶意函数名定义为 a，然后 a 参与后面的变换。通过这种变量替换的方式，原本直接通过函数名或者函数体来判断加密方式以及对应密文的方法变得不再可行。

字串拼接是指将一个完整的字串拆分成为几个字串，然后使用“+”连接符等方式将字串重新拼接成原文。例如下面的变量“YuTian123”的值是一个 activex 插件的名称字符串，new ActiveXObject(YuTian123)，则用于生成这个 ActiveX 插件对象提供给后面的调用。而这里名称字符串就是通过“+”连接符进行拼接。

字串替代指将明文中的若干内容使用预先定义的变量或者字串进行替换。字串替代需要与正则表达式的替换函数(例如 replace 等)或者字串拼接函数。

移位是指通过变量替换、字串替代、拼接等方式，以及编程语言中控制语句的使用，使原本简单的字符顺序就可以有很多不同的变化，一个样本就可以产生出多种不同的变种。

3.5 切分

这种方式是利用 HTML 所特有的，利用切分可以将恶意代码的不同部分放入不同的文件中，既可以妨碍检测和分析，也可以将公用部分作为其他恶意代码脚本的模块分离出来。利用〈script src="a. js"〉〈/script〉〈script src="b. js"〉〈/script〉这样的方式将不同功能放置到不同的 js 文件中，常见的可分离部分是漏洞利用脚本的 shellcode 部分，因为 shellcode 是可以被不同的漏洞利用脚本所公用的。

3.6 反检测技术

在 2008 年的 OWASP 上面，Wayne Huang[3] 已经对这方面进行了一定深度的介绍。例如环境检测、成员枚举方式获得函数接口、代码自校验等等。通过这些方式，可以干扰一些基于行为的检测。对于本文的检测方法，环境测试会有一定影响。

4 检测方法与实现

4.1 动态执行

在 Javascript 中由于动态执行函数的存在，才能使解密后的代码最终被执行。因此对于这些动态执行函数的监控是非常重要的，常见动态执行函数接口如表 1 所列。

表 1 常见动态执行函数接口

函数名称	对应功能
eval	执行字串代码
window. eval	执行字串代码
execScript	执行字串代码
window. execScript	执行字串代码
setTimeout	延迟后执行指定代码
setInterval	周期性的执行指定代码

对于恶意代码，在解密后一般都需要使用上述函数执行解密后的 Javascript 代码字串，因此对这些函数进行拦截，可以很好地对解密后的代码进行检测。不论恶意代码如何混淆，最终都必须通过解编码、解密等方法来获得明文，并通过 eval 等动态执行函数执行明文代码。利用这一点，本系统对 eval 等动态执行函数进行挂钩，并在这些地方对动态执行的代码进行检测，动态启发式设计如图 1 所示。

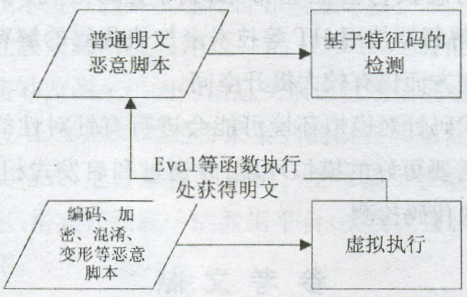

图 1 动态启发式设计

4.2 针对环境的检测

由于挂马代码常常放置在脚本中，最终通过 Document. write 等方式将挂马代码写入网页中，因此这种方式比直接在网页上加入挂马代码更具有隐蔽性。

多个连续的 document. write 函数写入的 HTML 文本是连续的，即 HTML 的挂马语句可以放入多个 document. write 函数中作为参数，对其功能实现没有影响。由于引擎对同一上下文代码使用同一环境，因此这样的代码也仍然能被引擎聚合并检测，常见动态写入网页的函数接口如表 2 所列。

表 2 常见动态写入网页的函数接口

函数	功能
document. write	向网页写入 HTML 文本
document. writeln	向网页写入 HTML 文本换行符结束

4.3 抗反检测技术

抗反检测技术的关键在于对环境的全面模拟，可以总结出一些关键的方法和属性。总结的一些相关方法和属性如表 3 所列。

表 3 Window 关键属性和方法

名称	功能
window. open	访问一个 URL
window. status	浏览器状态栏信息

由于环境中很多对象之间是相互引用的关系，因此这些对象之间的关联关系也必须模拟出来，如 document 和 window. document。

4.4 整体结构

针对恶意代码的现状，本文使用静态扫描和模拟执行相结合的方式，对未混淆变形的恶意代码可以使用静态特征码的方式检测，这种方法准确、可靠，并且速度较快。对于混淆脚本，则会通过动态执行的方式进行检测，并通过一定的条件进行判断，当触发条件时可以认为已经完成对恶意代码的解混淆，进而可以进入进一步扫描，即通过特征码方式检出恶意代码，整体结构如图 2 所示。

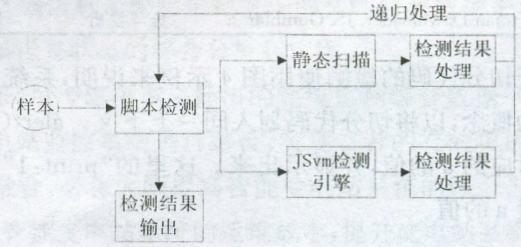

图 2 整体架构

之所以使用这样的结构，是因为网页中的 Javascript 代码不是孤立的存在，一段 Javascript 代码可以放在不同的 script 标签中，而这种拆分，浏览器是能够处理的，因为本引擎可以模拟环境使其处于同一个上下文环境中。

虚拟执行的恶意脚本检测部分可以分为若干处理模块。整个引擎可以大致划分为 JSVm，JSWindow 和 JS_Reactor，如图 3 所示。

JSVm 主要负责对 Javascript 脚本进行编译和运行，并保证整个运行机制的稳定和可靠，这部分使用 Spidermonkey 引擎作为基础，并在其基础上根据需要加入了相关运行时挂钩等，以便与检测等模块配合实现整体检测功能。

JSWindow 主要用于模拟宿主环境，例如常见的 window，doucument 等对象都进行相应的模拟，同时会在模拟环

境中加入检测点为检测模块提供接口。

JS_Reactor 用于进行各种运行时检测、环境检测以及启发式分析，在脚本运行完成后，还会对整个运行状态进行一次分析，将这些检测方法组织在一起，有利于检测更好地发挥整体优势，同时屏蔽不同模块之间的差异性，保证检测和其他模块的无缝结合。

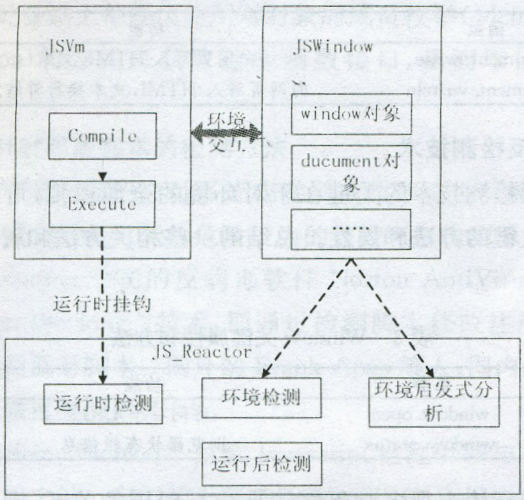

图 3　模拟执行模块结构

5　实验与测试结果

5.1　对混淆代码的检测

针对各种混淆类型的恶意代码，通过对实际样本的检测，均可以解密原有代码并进行检测，检测结果如表 4 所列。

表 4　检测结果

类型	样本名称	实际效果	运行时间
编码	Trojan-Downloader. HTML. IFrame. em	挂马语句检出	9ms
加密	Trojan-Downloader. JS. Agent. bra	网马检出	108ms
混淆	Exploit. HTML. Agent. by	挂马检出	25ms
变形	Trojan-Downloader. JS. Gumblar. a	挂马检出	11ms

对于切分代码的检测使用图 4 示例来说明，系统引入了上下文的概念，以将切分代码划入同一上下文。alert(a)就可以找到对应的 a 的值，并显示出来。这里的“print:1”即是打印出变量 a 的值。

```
<script>
a=1                          //script_1
</script>
<script>
alert(a)                     //script_2
</script>
```

```
html
script
script
print:1
```

图 4　切分代码测试结果

5.2　对反检测技术的实验

(1)抗“成员枚举方式获得函数接口”

图 5 的示例可以检出 h[i][j]为 document. write 接口，并检测其中内容。

```
<script>
h = this;
for (i in h)
{
//枚举全局对象，得到 document 对象
}
for (j in h[i])
{
//枚举 document 对象，得到 write 接口
}
}
h[i][j]("<html>test</html>");
</script>
```

```
document.write content:<html>test</html>
```

图 5　对成员枚举方式的检测实验

(2)函数自校验

针对 callee 的函数自校验方法，本系统是可以检出的。例如图 6 的示例代码，testCallee()函数可以正常执行，并由 document. write 打印出代码信息。

```
<script>
function testCallee(){return arguments.callee}
document.write(testCallee());
</script>
```

```
document.write content:function testCallee() {
    return arguments.callee;
}
```

图 6　对函数自校验的实验

(3)抗环境检测

利用环境检测逃过基于虚拟执行的恶意脚本检测对而言，比较简单。只要利用没有模拟到的接口，进行浏览器环境的测试，就可以确定是否是真实的浏览器环境，从而逃过基于虚拟执行的检测方式。例如 google 的 chrome 对象，绝大部分采用的是 ActiveX 对象。

结束语　本文通过对模拟执行的方式对编码、加密、混淆、变形、切分等形式的恶意代码检测有较好的改善效果，并对反检测技术有一定抵抗效果。

但由于使用动态模拟执行的检测方式，在此基础上可以对恶意代码的动态行为进行监控，并使用一些启发式规则达到更快速、高效的检测效果。

引入动态执行必然会带来时间上的损耗。但是当前 Javascript 解释器使用 JIT 等技术来加快代码的解释和执行，因此在速度方面仍有较大提升空间。

恶意代码针对模拟环境可能会进行有针对性的检测，针对这方面需要更好的模拟环境，并通过和启发式相结合的方式实现恶意代码检测。

参 考 文 献

[1]　Mozilla. JSAPI User Guide[EB/OL]. http://developer-stage.mozilla.org/En/SpiderMonkey/JSAPI_User_Guide

[2]　Nazario J. Reverse Engineering Malicious Javascript[J]. CanSecWest, 2007

[3]　Wang Huang. Web-based Malware obfuscation: the kung-fu and

the detection[R]. Taiwan:OWASP,2008
[4] Kolisar. WhiteSpace:A Different Approach to Javascript Obfuscation,DEFCON 16
[5] Symantec. What is Norton AntiVirus Script Blocking technology? [EB/OL]. http://service1. symantec. com/SUPPORT/nav. nsf/pfdocs/2001020912195106
[6] Apap F. Honing A, Hershkop S. Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses[C]// Fifth International Symposium on Recent Advances in Intrusion Detection(Zurich Switzerland,Oct 2002). 2002
[7] 鲍欣龙,等. 可用于恶意脚本识别的注册表异常行为检测技术[J]. 计算机工程,2005,31(8):137-139
[8] Feinstein B,Peck C. Caffeine monkey:Automated collection,detection and analysis of malicious Javascript[C]//Blackhat. 2007
[9] Crockford D. AdSafe: Making Javascript safe for advertising [EB/OL]. http://www. adsafe. org,2009
[10] Facebook, Inc. Fbjs [EB/OL]. http://wiki. developers. facebook. com/index. php/FBJS,2007
[11] Ajaxian. Facebook Javascript and security [EB/OL]. http://ajaxian. com/archives/facebook-Javascript-and-security, Aug. 2007
[12] Miller M S,Samuel M,Laurie B,et al. Caja:Safe active content in sanitized Javascript [EB/OL]. http://google-caja. google-code. com/files/caja-2007. pdf,2007
[13] Reis C,Dunagan J,Wang H,et al. BrowserShield:Vulnerability-driven filtering of dynamic HTML[C]// Proceedings of the Symposium on Operating Systems Design and Implementation. 2006
[14] Anderson C, Giannini P, Drossopoulou S. Towards type inference for Javascript[C]// Proc. 19th European Conference on Object-Oriented Programming. Glasgow,UK,July 2005:429-452
[15] Yu D,Chander A,Islam N,et al. Javascript Instrumentation for browser security[C]// Proceeding of Conference On Principles of Programming Language. Jan. 2007

(上接第 80 页)

设备;3)集中的在线监测主机。整个在线监测系统架构如图1所示(为了简化,图中仅例示了智能变电站中几类典型的电气设备)。

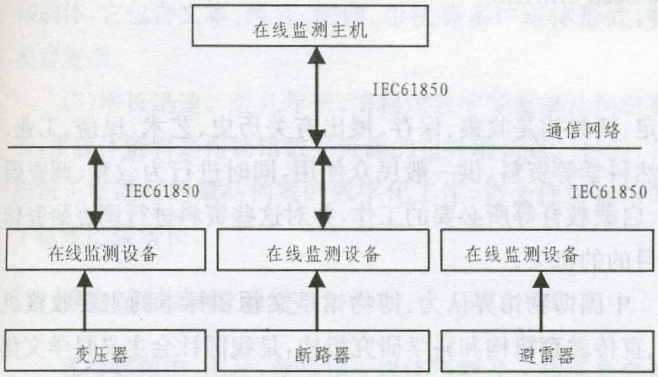

图 1 智能变电站在线监测系统架构

智能变电站在线监测系统采用分布式集中体系结构[5],各个电气设备的监测由分散的监测设备单独完成,各个设备具体负责数据的采集,并将采集到的设备状态信息发送至在线监测主机。在线监测主机根据搜集到的设备状态信息,构建统一的在线监测数据平台,结合监测设备的综合信息,基于高级诊断技术进行设备状态评价和风险评估。

整个在线监测系统采用统一的 IEC61850 通信规约。各个监测设备和监测主机间的信息交换通过通信网络完成。监测设备通过 IEC61850 规约将采集到的设备状态信息封装成标准的报文格式,上传至监测主机。主机基于收集到的设备和装置信息,搭建开放统一的数据平台,实现设备状态的综合分析和应用。

基于 IEC 61850 标准的在线监测系统架构,为构建变电站统一、开放的数据平台提供了可能。统一的数据平台形成后,系统中设备状态分析和评估所需的数据将直接从数据平台获取。同时,各设备间的信息交换也可通过数据平台上完成,从而实现了全站设备状态的整体评价和综合评估[1]。

结束语 传统的在线监测系统主要局限于单台设备、个别特征,难以对整个变电站中的设备进行多特征量的广泛监测、分析,无法对全站电气设备的状态进行整体评估。同时,现有的在线监测系统采用的私有或异构的通信标准,不能解决不同装置间的接口和互操作问题。因此,难以满足智能变电站在线监测功能的要求。

为了应对这些问题,本文提出了一种基于 IEC61850 的在线监测系统架构。该架构采用分布式集中体系结构,通过通信网络,将不同的监测装置和在线监测主机连接在一起。各监测装置负责设备状态信息的采集;后台主机根据接收到的设备状态信息来构建全站统一的在线监测数据平台,从而实现全站设备状态的综合分析和应用。

未来,可考虑将本文提出的在线监测综合数据平台与现有的变电站监控系统进行融合,实现二者的有效集成。通过集成和融合,必将大幅提高智能变电站系统的综合自动化水平,降低智能变电站系统的运维成本,提升变电站系统的智能化水平,从而推动智能电网技术的发展。

参 考 文 献

[1] 徐清超. 变电站在线监测技术的发展方向[J]. 红水河,2009,28(5):71-75
[2] 吉亚民,谢林枫. 江苏电网电气设备在线监测平台建设的研究[J]. 江苏电机工程,2009,28(5):36-38
[3] 鲁国光,刘骥,张长银. 变电站的数字化技术发展[J]. 电网技术,2006,30(Supp.):499-504
[4] 何晓英. IEC61850 标准数字变电站的建设与发展[J]. 陕西电力,2006,34(4)
[5] 周建国,李红雷,赵文彬. 基于数字化平台的输变电设备在线分析和状态维修技术[J]. 华东电力,2009,37(7):1075-1079