MỘT SỐ VẤN ĐỀ AN TOÀN CHO CÁC ỨNG DỤNG TRÊN NỀN WEB

Nguyễn Sỹ Hòa[a], Lại Thị Nhung[b], Đặng Thanh Hải[c*]

[a]Khoa Công nghệ Thông tin và Truyền thông, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội, Hà Nội, Việt Nam
[b]Khoa Khoa học Cơ bản, Trường Đại học Điều dưỡng Nam Định, Nam Định, Việt Nam
[c]Khoa Công nghệ Thông tin, Trường Đại học Đà Lạt, Lâm Đồng, Việt Nam

Lịch sử bài báo
Nhận ngày 10 tháng 03 năm 2017 | Chỉnh sửa ngày 04 tháng 05 năm 2017
Chấp nhận đăng ngày 19 tháng 05 năm 2017

Tóm tắt

Internet đã và đang phát triển vô cùng mạnh mẽ, kết nối hơn 30% dân số thế giới, với hơn 2.2 tỷ người dùng. Lợi ích mà Internet đem lại cũng đi kèm với các rủi ro: Năm 2015 toàn thế giới thiệt hại hơn 445 tỷ USD, Việt nam thiệt hại khoảng 8700 tỷ VNĐ do các sự cố về tấn công mạng; 5226 trang web của các cơ quan doanh nghiệp tại Việt Nam bị tấn công. Nguyên nhân chính đó là sự tiến bộ của kẻ tấn công, sự ra đời của những công nghệ mới, và cũng vì sự thiết lập của những hệ thống ngày nay cũng ngày càng phức tạp và khó quản lí hết rủi ro. Vì vậy, việc nghiên cứu đánh giá những nguy cơ có tính rủi ro cao đối với các ứng dụng dựa trên nền web là nhu cầu cấp thiết đối với các cơ quan, tổ chức đang triển khai ứng dụng web trên Internet. Trong bài báo này, chúng tôi sẽ phân tích những lỗ hổng phổ biến nhất của các ứng dụng trên nền web và khuyến nghị các phương pháp để kiểm tra phát hiện.

Từ khóa: An toàn ứng dụng web; Bảo mật; Ứng dụng web.

1. GIỚI THIỆU

Internet đã và đang phát triển vô cùng mạnh mẽ, kết nối hơn 30% dân số thế giới, với hơn 2.2 tỷ người dùng. Việc kinh doanh truyền thống dần được thay thế bằng các giao dịch điện tử hay thương mại điện tử (TMĐT) mà phần lớn dựa vào các ứng dụng trên nền web. Theo báo cáo năm 2015 của Bộ Công thương, TMĐT của Việt Nam trong năm 2015 đã tăng 37% so với năm 2014, đạt khoảng 4.07 tỷ USD, chiếm khoảng 2.8% tổng mức bán lẻ hàng hóa và doanh thu dịch vụ tiêu dùng cả nước. Theo báo cáo của eMarketer, doanh thu TMĐT bán lẻ của Mỹ năm 2015 ước đạt khoảng 355 tỷ USD, chiếm khoảng 7.4% tổng doanh thu bán lẻ của nước này. Doanh thu bán lẻ trực tuyến TMĐT của Trung

* Tác giả liên hệ: Email: haidt@dlu.edu.vn

Quốc tính đến tháng 9/2015 ước đạt 672 tỷ USD, tăng 42.1% so với cùng kỳ năm 2014 và chiếm khoảng 15.9% tổng doanh thu bán lẻ. Còn tại Hàn Quốc, doanh thu bán lẻ TMĐT của thị trường này trong năm 2015 là 38.86 tỷ USD, chiếm 11.2% tổng doanh thu bán lẻ. Liên quan đến dự báo về quy mô của thị trường TMĐT thời gian tới, hồi cuối tháng 1/2016, đại diện lãnh đạo Hiệp hội TMĐT Việt Nam (VECOM) đã cho biết, cuối năm 2015, hãng nghiên cứu thị trường Ken Research đã đưa ra dự đoán quy mô thị trường bán lẻ Việt Nam năm 2019 sẽ đạt 7.5 tỷ USD.

Song song với các lợi ích mà Internet mang lại, cũng có những rủi ro như: Năm 2015 toàn thế giới thiệt hại hơn 445 tỷ USD, Việt nam thiệt hại khoảng 8700 tỷ VNĐ do các sự cố về tấn công mạng; 5226 trang web của các cơ quan doanh nghiệp tại Việt Nam bị tấn công. Nguyên nhân chính đó là sự tiến bộ của kẻ tấn công, sự ra đời của những công nghệ mới, và cũng vì sự thiết lập của những hệ thống ngày nay cũng ngày càng phức tạp và khó quản lí hết rủi ro. Vì vậy, việc nghiên cứu đánh giá những nguy cơ có tính rủi ro cao đối với các ứng dụng dựa trên nền web là nhu cầu cấp thiết đối với các cơ quan, tổ chức đang triển khai ứng dụng web trên Internet. Trong bài báo này, chúng tôi sẽ phân tích những lỗ hổng phổ biến nhất của các ứng dụng trên nền web và khuyến nghị các phương pháp để kiểm tra phát hiện.

2. NHỮNG LỖI THƯỜNG GẶP TRONG CÁC ỨNG DỤNG WEB

Khi một ứng dụng web (có thể là Website hoặc WebApp) được triển khai trên Internet, nó trở thành mục tiêu phá hoại của những kẻ tấn công muốn tìm và khai thác những lỗ hổng bảo mật xuất hiện trong ứng dụng. Dưới đây là những lỗ hổng bảo mật mà người phát triển các ứng dụng web cần tránh khi phát triển ứng dụng.

2.1. Tấn công lỗi truy vấn SQL

Theo Data (2016), tấn công lỗi truy vấn SQL sử dụng chuỗi lệnh ngôn ngữ truy vấn có cấu trúc (SQL) để truy vấn trực tiếp cơ sở dữ liệu. Ứng dụng thường sử dụng lệnh SQL để xác thực người dùng với ứng dụng xác nhận vai trò và mức độ truy cập, lưu giữ và lấy thông tin cho ứng dụng và người dùng, và các đường dẫn tới nguồn dữ liệu. Sử dụng lỗi truy vấn SQL, những kẻ tấn công có thể sử dụng một ứng dụng web dễ bị tổn

thương để tránh các biện pháp bảo mật thông thường và truy cập trực tiếp đến dữ liệu có giá trị.

Lý do tấn công lỗi truy vấn SQL làm việc là do ứng dụng không xác nhận đầu vào một cách phù hợp trước khi qua nó đến lệnh SQL. Tấn công lỗi truy vấn SQL có thể vào qua thanh địa chỉ, trong trường ứng dụng và qua câu truy vấn và tìm kiếm. Tấn công SQL có thể cho phép kẻ tấn công: Đăng nhập vào ứng dụng mà không cần cung cấp chứng nhận hợp lệ; Thực hiện những câu truy vấn dữ liệu trong cơ sở dữ liệu mà ứng dụng thường không truy cập đến những dữ liệu đó; Thay đổi nội dung cơ sở dữ liệu hoặc thả vào các cơ sở dữ liệu; Sử dụng mối quan hệ tin cậy được thiết lập giữa các thành phần ứng dụng web để truy cập các cơ sở dữ liệu khác. Ví dụ: Trên các ứng dụng Web, các trang web thường có các ô nhập văn bản để người dùng nhập thông tin vào cho các câu lệnh SQL thực hiện trên Server để truy vấn dữ liệu. Thông tin vào quyết định cách ứng dụng xử lý. Tuy nhiên, không phải trình ứng dụng nào cũng có đầy đủ cơ chế, chính sách đảm bảo an toàn. Lợi dụng lỗi về an toàn của các ứng dụng chưa được kiểm soát hết, kẻ tấn công có thể chèn vào các ô nhập văn bản các đoạn mã độc thay vì các thông tin chuẩn mực để ứng dụng xử lý. Chẳng hạn, một câu lệnh SQL với thông tin đầu vào từ ô nhập văn bản như sau:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") +"'";
Exec(query);
```

Câu lệnh trên đơn thuần chỉ là đưa ra các thông tin về tài khoản của một người dùng với mã là id. Tuy nhiên kẻ tấn công có thể thay thế tham số ‘id’ cần nhập vào bằng chuỗi sau: *‘ or ‘1’=’1*.

Câu lệnh truy vấn trên Server trở thành:

```
String query = "SELECT * FROM accounts WHERECustID='" + request.getParameter ("‘ or ‘1’=’1") +"'";
Exec(query);
```

Việc này thay đổi ý nghĩa của câu truy vấn là trả lại giá trị của tất cả các tài khoản trong cơ sở dữ liệu thay vì chỉ của một người dùng. Trong trường hợp xấu nhất, kẻ tấn

công có thể sử dụng các thông tin đánh cắp được để thực thi những thủ tục lưu trữ trong cơ sở dữ liệu và giúp chiếm quyền điều khiển cơ sở dữ liệu.

2.2. Thực thi mã script xấu

Theo Data (2016) thì thực thi mã scrip xấu được biết với tên là Cross-Site Scripting (XSS) dựa trên việc mã hóa thiếu những kí tự đặc biệt (ví dụ như >, <) trên các trang Web. Lợi dụng điều này, kẻ tấn có thể chèn một đoạn script trong đầu vào chẳng hạn "<script src='http://evil.com'; ></script>". Với cách làm này, khi truy cập đến trang web nào, trang web đó sẽ tải một tập tin dạng script từ một trang web bên ngoài mà sẽ gửi cookies hoặc dữ liệu cục bộ của người sử dụng đến kẻ tấn công.

Để ngăn chặn XSS, người phát triển ứng dụng web cần phải mã hóa tất cả nội dung xuất hiện trên một trang web. Cụ thể, người phát triển cần mã hóa không chỉ nội dung, mà cần mã hóa cả URL và các thuộc tính. Nhiều nền tảng (*framework*) phát triển web hỗ trợ sẵn một phần tính năng mã hóa này. Trong ASP.NET MVC, Razor View Engine sẽ tự động mã hóa tất cả văn bản và giá trị thông qua việc sử dụng Model và các hàm HTML Helpers. Ở những chỗ không hỗ trợ khả năng mã hóa, người phát triển có thể sử dụng hàm *html.Encode*. Đối với giá trị *url* và các thuộc tính, người phát triển cần phải tự gọi các hàm *url.Encode* và *html.AttributeEncode* để mã hóa. Mặc dù vậy, mã hóa HTML là không đủ để bảo đảm trang web an toàn trước tấn công XSS. Kẻ tấn công có thể chèn các script vào nội dung của Javascript có trên trang web. Vì vậy, chúng ta cũng phải mã hóa nội dung có trong script (Hình 1).

```
@section scripts {
  @if (ViewBag.UserName != null)  {
    <scripttype="text/javascript">
      $(function () {
        var msg = 'Welcome, @Ajax.JavaScriptStringEncode(ViewBag.UserName)!';
        $("#welcome-message").html(msg).hide().show('slow');
      });
    </script>  }}
```

Hình 1. Sử dụng hàm *html.Encode*

Trong ASP.Net MVC, chúng ta có thể sử dụng hàm *Ajax.JavaScriptString Encode* để mã hóa những chuỗi được sử dụng trong Javascript tương tự như sử dụng *html.Encode*, ví dụ được thể hiện như trong Hình 1.

Một giải pháp khác để mã hóa HTML và Javascript là sử dụng thư viện AntiXSS.AntiXSS giúp thay thế mã hóa ngầm định (*default encoder*) được sử dụng trong ASP.Net và do đó chịu trách nhiệm mã hóa nội dụng khi sử dụng các hàm Encode Helper được nói ở trên. Riêng đối với mã hóa Javascript, thay vì sử dụng hàm *Ajax.JavaScriptStringEncode*, người phát triển cần sử dụng hàm *Encoder.JavaScript Encode* như trong Hình 2.

```
@using Microsoft.Security.Application
@section scripts {
   @if (ViewBag.UserName != null)
   {
      <scripttype="text/javascript">
         $(function () {
       var msg = 'Welcome, @Encoder.JavaScriptEncode(ViewBag.UserName, false)!';
            $("#welcome-message").html(msg).hide().show('slow');
         });
      </script> }}
```

Hình 2. Sử dụng hàm *Encoder.JavaScriptEncode*

2.3. Tấn công giả mạo yêu cầu

Tấn công giả mạo yêu cầu Cross-Site Request Forgery (CSRF, XSRF) (Veracode, 2017) hay nói cách khác là làm sai lệch bản chất của một yêu cầu (*request*) mà người dùng (hay chính xác hơn là các trình duyệt) không thể nhận biết được. Ví dụ, giả sử trên một trang web có sử dụng thẻ img để tải hình ảnh như sau:

![””/](”/picture.jpg” alt=””/)

Vì một lý do nào đó (do bị hack hay do sự nhầm lẫn của người phát triển) thẻ img có thể bị biến đổi thành:

![””/](”/account/logout” alt=””/)

Khi đó, nếu trang này được tải, trình duyệt sẽ gửi một yêu cầu tới Server với mục đích là tải hình ảnh nhưng thực chất lại khiến cho Server thực thi lệnh thoát người dùng. Đây chính là một dạng của CSRF.

Trong thực tế, tấn công CSRF thường ở dạng những đường liên kết được gửi từ Spammer tới Email hoặc các Website xã hội mà nếu người sử dụng bất cẩn click vào sẽ bị chuyển hướng tới một trang web khác mà trang này có thể gửi các thông tin của người sử dụng tới những nơi mà kẻ tấn công muốn. Khi đó, về mặt lý thuyết, chính người sử dụng đó đã gửi thông tin nhưng trên thực tế thì người sử dụng không hề hay biết.

Để ngăn chặn CSRF, chúng ta cần một cơ chế xác định có phải chính người sử dụng muốn thực hiện yêu cầu đó không. ASP.NET MVC cung cấp một giải pháp khá tốt cho vấn đề này gọi là kiểm tra Token (*Token verification*). Đầu tiên, người phát triển ứng dụng sẽ sử dụng hàm *Html.AntiForgeryToken* để sinh ra một trường ẩn nằm trong biểu mẫu mà chứa một giá trị bất kỳ duy nhất. Giá trị này cũng sẽ được lưu trong một Session Cookie ở phía trình duyệt của máy khác. Khi biểu mẫu được gửi, hai giá trị này sẽ được so sánh nhờ vào Action Filter Validate Antiforgery Token.

```
public class IsPostedFromThisSiteAttribute: AuthorizeAttribute
{
   public override void OnAuthorize(AuthorizationContext filterContext)
   {
      if (filterContext.HttpContext != null)
      {
         if (filterContext.HttpContext.Request.UrlReferrer == null)
            thrownew System.Web.HttpException("Invalid submission");
 if (filterContext.HttpContext.Request.UrlReferrer.Host != "trandev90.com.com")
 thrownew System.Web.HttpException("This form wasn't submitted from this site!");
      }
   }
}
```

Hình 3. Thực thi *Code action method*

Giải pháp *Token verification* sẽ xử lý được hầu hết các tấn công CSRF thông thường, nhưng không phải tất cả. Một giải pháp khác tương tự *Token verification* nhưng thay vì kiểm tra Token tự sinh ra thì sẽ kiểm tra thuộc tính *HttpReferrer* của yêu cầu để đảm bảo yêu cầu đến từ trang web được sinh ra bởi chính Website đang nhận yêu cầu chứ

không phải đến từ một trang web được sinh ra bởi Website. Giải pháp này gọi là *HttpReferrer Validation*. Người phát triển có thể cài đặt kiểm tra trong một bộ lọc mà sẽ được thực thi trước khi thực thi *Code action method* như trong Hình 3. Một quy tắc giúp giảm nhẹ khả năng tấn công CSRF là luôn xử lý những thay đổi dữ liệu trong cơ sở dữ liệu hay trong Website dưới dạng yêu cầu POST.

2.4. Bảo vệ lớp truyền dẫn không hiệu quả

Theo Veracode (2017) thì bảo vệ lớp truyền dẫn không hiệu quả có thể cho phép các bên thứ ba không đáng tin cậy được truy cập trái phép thông tin nhạy cảm. Thông tin liên lạc giữa các trang web và khách hàng nên được mã hóa hoặc dữ liệu có thể bị chặn, trích, hoặc chuyển hướng. Các mối đe dọa khác nhau như trộm cắp tài khoản, tấn công lừa đảo, và tài khoản quản trị có thể xảy ra sau khi hệ thống đang được cho phép truy cập. SSL/ TLS nên được sử dụng để xác thực trên các trang web.

2.5. Lưu trữ mật mã thiếu an toàn

Các ứng dụng web sử dụng thuật toán mã hóa để mã hóa dữ liệu và thông tin nhạy cảm khác được chuyển từ máy chủ cho khách hàng hoặc ngược lại. Lưu trữ mật mã thiếu an toàn (The OWASP Foundation, 2017a) đề cập đến trạng thái của một ứng dụng với mã hóa yếu được sử dụng để lưu trữ an toàn dữ liệu trong cơ sở dữ liệu. Vì vậy, các dữ liệu không an toàn có thể dễ dàng bị tấn công và sửa đổi bởi các đối tượng tấn công để có được thông tin bí mật và nhạy cảm như thông tin thẻ tín dụng, mật khẩu, số SSN, và thông tin xác thực khác với mã hóa để khởi động trộm cắp danh tính, gian lận thẻ tín dụng, và các tội khác. Các nhà phát triển có thể tránh được các cuộc tấn công bằng cách sử dụng các thuật toán thích hợp để mã hóa các dữ liệu nhạy cảm.

2.6. Việc quản lý xác thực và phiên làm việc bị đứt quãng

Quản lý xác thực và phiên làm việc (The OWASP Foundation, 2017b) bao gồm tất cả các khía cạnh của việc xác thực người sử dụng và quản lý các phiên làm việc đang hoạt động. Tuy nhiên việc xác thực không thành công là do các chức năng biểu thị kém như thay đổi mật khẩu, quên mật khẩu, ghi nhớ mật khẩu, cập nhật tài khoản… Phải đặc

biệt chú ý đến việc xác thực người sử dụng. Sẽ tốt hơn khi sử dụng các phương pháp xác thực cao thông qua những Token hoặc sinh trắc học về mật mã dựa trên phần cứng và phần mềm đặc biệt. Kẻ tấn công dựa vào những điểm yếu trong các chức năng quản lý xác thực hoặc phiên làm việc như các tài khoản hiển thị, ID phiên làm việc, thoát ra, quản lý mật khẩu, thời gian tạm ngưng, ghi nhớ, câu hỏi bí mật, cập nhật tài khoản và các chức năng khác để đóng giả người sử dụng. Các kiểu tấn công thường gặp là:

- *Trộm ID phiên làm việc trong URL*: Kẻ tấn công xen vào đường liên lạc của mạng lưới hoặc lừa người sử dụng để lấy ID phiên làm việc và sử dụng ID phiên làm việc với mục đích phá hoại;

- *Khai thác thời gian tạm ngưng*: Nếu thời gian tạm ngưng của chương trình ứng dụng không được thiết lập phù hợp và đóng trình duyệt một cách đơn giản mà không cần thoát ra khỏi các trang đã truy cập trên một máy tính công cộng thì khi đó kẻ tấn công có thể sử dụng trình duyệt đó và khai thác các quyền của người sử dụng;

- *Khai thác mật khẩu:* Kẻ tấn công truy cập vào cơ sở dữ liệu mật khẩu của ứng dụng web. Nếu mật khẩu của người sử dụng không được mã hóa, kẻ tấn công có thể khai thác tất cả mật khẩu của người sử dụng.

2.7. Điều hướng không mong đợi

Bất kỳ ứng dụng web nào mà điều hướng (*redirect*) tới một URL được mô tả qua câu truy vấn hay biểu mẫu đều có thể bị kẻ tấn công khai thác để điều hướng người sử dụng tới một Website bên ngoài mà người sử dụng lầm tưởng là của Website hiện tại. Cách thức tấn công này gọi là chuyển hướng không mong đợi hay chuyển hướng mở (*open redirection*) (CodeProject, 2017). Rõ ràng, chúng ta dễ nhận biết là tham số *ReturnUrl* được bao gồm trong câu truy vấn khi người sử dụng được chuyển tới trang đăng nhập trong ASP.NET MVC. Nếu tham số này bị thay đổi bởi kẻ tấn công (có thể dưới dạng những liên kết mà kẻ tấn công gửi đến người sử dụng) và phía máy chủ không kiểm tra tham số này thì khi người sử dụng đăng nhập thành công sẽ được chuyển tới tới

Website của kẻ tấn công giả mạo Website hiện tại để yêu cầu người sử dụng nhập lại tên và tài khoản cũng như nhập thêm thông tin khác.

ASP.NET MVC cung cấp một hàm để kiểm tra URL tên là *Url.IsLocalUrl* và cũng sinh ra một đoạn mã để kiểm tra *returnURL* sau khi người sử dụng đăng nhập thành công như trong Hình 4.

```
private ActionResult RedirectToLocal(string returnUrl)
{
        if (Url.IsLocalUrl(returnUrl))
        {
                return Redirect(returnUrl);
        }
        else
        {
                return RedirectToAction("Index", "Home");
        }
}
```

Hình 4. Sử dụng hàm *Url.IsLocalUrl*

2.8. Sai sót cấu hình an ninh

Các lập trình viên và quản trị mạng nên kiểm tra để đảm bảo toàn bộ ngăn xếp được thiết lập một cách hợp lý nếu không dẫn đến việc sai sót cấu hình an ninh (Hackingstuffs, 2017) có thể xảy ra ở bất cứ cấp độ nào của một ngăn xếp ứng dụng, bao gồm nền hệ thống, máy chủ web, máy chủ ứng dụng, chương trình khung và mã tùy chọn. Ví dụ, nếu máy chủ không được thiết lập một cách hợp lý sẽ dẫn đến kết quả là có rất nhiều rắc rối có thể ảnh hưởng đến an ninh của một trang web. Các rắc rối này dẫn đến những hậu quả như gây nên lỗ hổng phần mềm máy chủ, lỗ hổng bảo mật chưa được vá, cho phép các dịch vụ không cần thiết và tính xác thực không hợp lý. Một vài vấn đề có thể được phát hiện dễ dàng với sự giúp đỡ của máy quét tự động. Những kẻ tấn công có thể truy cập vào các tài khoản mặc định, các trang web không được sử dụng, những lỗ hổng không được vá, những tập tin và thư mục không được bảo vệ… để chiếm được quyền truy cập hợp pháp. Tất cả những tính năng không cần thiết và không an toàn phải được bảo vệ và sẽ rất có lợi nếu chúng bị vô hiệu hóa hoàn toàn, do vậy những kẻ bên ngoài không thể lợi dụng chúng để tiến hành những cuộc tấn công độc hại. Mọi tập tin dựa trên các ứng dụng phải được quan tâm về tính xác thực phù hợp và các phương pháp

bảo mật mạnh mẽ nếu không những thông tin quan trọng có thể bị rò rỉ tới những kẻ tấn công.

Những tính năng không cần thiết nên được vô hiệu hóa hoặc thay đổi gồm:

- Giao diện điều khiển quản trị của máy chủ được tự động cài đặt và không được dỡ bỏ;
- Tài khoản mặc định không được thay đổi;
- Những kẻ tấn công phát hiện ra trang quản trị chuẩn trên Server, đăng nhập bằng mật khẩu mặc định và tiếp quản.

2.9. Đăng tin rác

Đăng tin rác (*Over-posting*) (Codedx, 2017) xảy ra trong những nền tảng web sử dụng tính năng liên kết các thuộc tính (*model binding*), chẳng hạn như trong ASP.NET MVC. Khi một người dùng sử dụng chế độ để trả lại HTML nhưng không trả lại hết các thuộc tính sẽ tạo cơ hội cho kẻ tấn công sử dụng các công cụ phát triển web để chèn thêm các giá trị không mong muốn vào các thuộc tính chưa dùng của chế độ đó.

Cách đơn giản để ngăn tình trạng này xảy ra là sử dụng thuộc tính *Bind* để xác định tường minh thuộc tính nào trong chế độ sẽ được liên kết. Thuộc tính *Bind* có thể được đặt trên mô hình lớp (*class*) hoặc trong các tham số của phương thức hoạt động như trong Hình 5.

```
[Bind(Include=”Name, Comment”)]
public class Review
{
        publicint ReviewID { get; set; }
        publicstring Name { get; set; }
        publicstring Comment { get; set; }
        publicbool Approved { get; set; }
}
```

Hình 5. Mô hình lớp Review

Một cách khác là sử dụng hàm *UpdateModel* hoặc *TryUpdateModel* như sau:

UpdateModel(review, “Review”, newstring[] { “Name”, “Comment” });

2.10. Trộm cắp Cookie

Các Website sử dụng Cookies để lưu thông tin người sử dụng giữa các yêu cầu. Cookies có thể chứa những thông tin không quan trọng như sự ưa thích hay lịch sử hoặc có thể lưu những thông tin rất quan trọng ví dụ như xác thực. Có hai loại Cookies:

- *Session Cookie*: Những Cookies chứa trong bộ nhớ của trình duyệt và sẽ bị mất đi khi tắt tắt trình duyệt hoặc hết phiên làm việc;
- *Persistent Cookies*: Những Cookies được lưu trữ trong các tệp văn bản trên ổ cứng của máy người sử dụng và sẽ không bị mất đi trừ khi người sử dụng xóa Cache của trình duyệt.

Cùng với sự tiện lợi thì Cookies cũng là cơ hội để kẻ tấn công sử dụng XSS để ăn cắp Cookies (Hackingstuffs, 2017) của người sử dụng và sau đó sử dụng nó để giả danh người đó thực hiện các thao tác khác. Bởi vì XSS có thể dùng script để có thể ăn cắp Cookies nên một cách hiệu quả để chống mất cắp Cookies là không cho phép script truy xuất Cookies bằng cách sử dụng HttpOnly.

Để ngăn chặn mã truy xuất lên tất cả Cookies của Website, nên đặt cấu hình trong web.config như sau:

```
<system.web>
    <httpCookiesdomain=””requireSSL=”false”httpOnlyCookies=”true”/>
</system.web>
```

Để thiết lập *HttpOnly* lên từng Cookie có thể sử dụng mã khi ghi Cookies như sau:

```
Response.Cookies[“MyCookie”].Value= “Remembering you…”;

Response.Cookies[“MyCookie”].HttpOnly = true;
```

3. CÁC CÔNG CỤ BẢO MẬT ỨNG DỤNG WEB

Phần này sẽ giới thiệu các công cụ quyét lỗ hổng trên web phổ biến nhất hiện nay (*Web Vulnerability Scanner*).

3.1. Nikto

Là một phần mềm mã nguồn mở với tính năng Web Server Scanner, tính năng kiểm tra các máy chủ web. Bao gồm hơn 3200 phương thức nhận diện các tập tin, lỗi nguy hiểm, hỗ trợ hơn 625 phiên bản WebServer, bao gồm những lỗi trên 230 WebServer khác nhau. Tính năng Scan kết hợp với các Plugins luôn được cập nhật tự động đảm bảo đưa ra kết quả đầy đủ và chính xác nhất. Là một công cụ rất hữu hiệu nhưng không được cập nhật thường xuyên. Các lỗi mới nhất thường được cập nhập chậm và không thể tìm thấy.

3.2. Paros Proxy

Một ứng dụng kiểm tra các lỗ hổng bảo mật trên các ứng dụng WebProxy. Một trang web trên nền Java thường kết hợp dạng ủy quyền (*proxy*) điều đó dẫn tới có nhiều lỗ hổng bảo mật. Phần mềm này hỗ trợ cho phép thay đổi/xem các gói tin HTTP/HTTPS và thay đổi chúng ở Cookies. Bao gồm một tính năng Web Recorder, Web Spider, và công cụ Scanner cho phép kiểm tra các ứng dụng có khả năng bị tấn công như lỗi SQL Injection và Cross-site Scripting.

3.3. WebScarab

Sản phẩm này cho phép phân tích các ứng dụng giao tiếp sử dụng giao thức HTTP và HTTPS. Một dạng rất đơn giản, WebScarab lưu các yêu cầu và trả lời cho phép tái sử dụng trong các phiên làm việc khác. WebScarab được thiết kế cho mọi người muốn xem hoạt động của các ứng dụng sử dụng giao thức HTTP(S), tuy nhiên công cụ này cũng cho phép phát triển và sửa những lỗi khó, hoặc cho phép nhận biết những lỗ hổng bảo mật trên các ứng dụng được thiết kế và triển khai.

3.4. WebInspect

Một công cụ tìm kiếm lỗ hổng trên ứng dụng web rất hiệu quả. SPI Dynamics' WebInspect với những công cụ trợ giúp người dùng nhận dạng ra những lỗ hổng bảo mật trên ứng dụng Web. WebInspect có thể giúp kiểm tra các web Server cấu hình đã chuẩn chưa, và cố gắng thử một vài dạng tấn công như Prameter Injection, Cross-site Scripting, Directory Traversal.

3.5. Whisker/Libwhisker

Một công cụ với thư viện mở rất phong phú giúp tìm kiếm các lỗ hổng bảo mật trên ứng dụng web. Libwhisker một tính năng của ngôn ngữ Perl – Module cho phép kiểm tra ứng dụng HTTP. Cung cấp những tính năng giúp kiểm tra máy chủ HTTP bằng cách lấy những sự hiểu biết về bảo mật, những nguy cơ tấn công để kiểm tra các lỗ hổng bảo mật. Whisker là một công cụ tìm kiếm lỗ hổng bảo mật sử dụng Libwhisker nhưng giờ có công cụ khác hiệu quả hơn đó là Nikto cũng sử dụng Libwhisker.

3.6. Burpsuite

Kết hợp với ứng dụng để tấn công ứng dụng web. Burp cho phép một kẻ tấn công tích hợp nhiều thao tác bằng tay hay những phương pháp tự động để tấn công, phân tích và khai thác các lỗ hổng bảo mật trên ứng dụng web.

3.7. Wikto

Một công cụ dùng để đánh giá mức độ bảo mật của máy chủ web. Là một công cụ dùng để kiểm tra các thiếu sót khi cấu hình máy chủ WebServer. Cho phép cung cấp nhiều mô đun khác nhau (nếu tích hợp thêm sẽ phải tốn phí), ví dụ như Back-End Function hay Google Integration. Wikto được viết bằng công nghệ .NET của Microsoft, để tải về mã nguồn của phần mềm này bạn phải đăng ký trên Website.

3.8. Acunetix Web Vulnerability Scanner

Một phiên bản thương mại của chương trình tìm kiếm các lỗ hổng bảo mật trên các ứng dụng Web. Acunetix WVS tự động kiểm tra các ứng dụng web để tìm kiếm các

lỗ hổng bảo mật như SQL Injection, hay Cross-Site Scripting, tìm kiếm những chính sách đối với mật khẩu đăng nhập cũng như các phương thức xác thực vào Website. Với giao diện đồ họa thân thiện, những báo cáo đầy đủ cho phép kiểm tra những vấn đề trên máy chủ và ứng dụng web.

3.9. Watchfire AppScan

Một phiên bản thương mại của chương trình tìm kiếm các lỗ hổng bảo mật trên các ứng dụng Web. AppScan cho phép kiểm tra những ứng dụng được phát triển trên nền web, dễ dàng kiểm tra và phát hiện lỗ hổng. AppScan kiểm tra nhiều lỗ hổng bảo mật, như Cross-Site Scripting, HTTP Response Spliting, và một vài dạng tấn công phổ biến khác, phát hiện các Trojan và Backdoor đang tồn tại trên máy chủ web và nhiều hơn nữa.

3.10. N-Stealth

Là một phiên bản thương mại, ứng dụng cho việc tìm kiếm các lỗ hổng bảo mật trên máy chủ Web. Phần mềm tự động cập nhật thường xuyên hơn các phần mềm miễn phí như Whisker/Libwhisker hay Nikto, nhưng nhiều lỗi mới trên web cũng không phát hiện kịp thời và nhanh chóng. Phần mềm bao gồm hơn 30.000 lỗ hổng có thể quét và khai thác trực tiếp, cùng với rất nhiều những cập nhật hàng ngày. Dễ dàng triển khai kết hợp với những phương pháp bảo vệ lỗ hổng bảo mật khác như: Nessus, ISS Internet Scanner, Retina, SAINT và Sara, bao gồm các tính năng khác. N-Sealth là phiên bản chỉ dành riêng cho Windows và không thể tải về mã nguồn. Gần đây nhóm CDC (Hacker Cult of the Dead Cow) đã tạo một công cụ rất hay cho việc kiếm và lấy thông tin nhạy cảm. Công cụ đó có tên gọi là Goolad Scan, là phần mềm dễ dùng đến nỗi người không hiểu biết gì về mã lập trình cũng có thể dò tìm những dữ liệu này để khai thác. Goolad Scan sẽ dẫn người quản trị hệ thống Website cài đặt nó để sau đó kẻ tấn công truy nhập hệ thống một cách dễ dàng. CDC cho biết họ đã thử nghiệm và thấy được những lỗ hổng ở các máy chủ tại Bắc Mỹ, châu Âu và Trung Đông.

4. KẾT LUẬN

Trong bài báo này chúng tôi đã hệ thống hóa các lỗi thường gặp trong các ứng dụng web và chỉ ra các cách khắc phục. Tuy nhiên, ngoài những lỗi thường gặp này, một ứng dụng web cũng có thể tồn tại rất nhiều các lỗi khác. Vì vậy, chúng tôi cũng đã trình bày các công cụ để có thể quét các lỗi của ứng dụng web. Mặc dù vậy, không một công cụ nào có thể phát hiện được tất cả các lỗi, mỗi một công cụ có điểm mạnh và yếu khác nhau. Do đó, người dùng nên sử dụng đồng thời nhiều công cụ để thực hiện kiểm tra tính an toàn cho một trang web của mình.

LỜI CẢM ƠN

Bài báo này được hoàn thành bởi sự hỗ trợ kinh phí từ đề tài TN.16.04, Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội.

TÀI LIỆU THAM KHẢO

Bavisi, J. (2016). *Certified ethical hacking certification*. Retrieved from https://www.eccouncil.org/programs/certified-ethical-hacker-ceh/

Codedx. (2017). *Security misconfiguration*. Retrieved from http://codedx.com/security-misconfigura-tion/

CodeProject. (2017). *Developing secure web applications: XSS attack, the confused deputy and over-posting*. Retrieved from http://www.codeproject.com/Tips/845612/Developing-Secure-Web-Applications-XSS-Attack-the/

Data, R. (2016). *SQL injection*. Retrieved from https://www.w3schools.com/sql/sql_injection.asp/

Hackingstuffs. (2017). *Cookies Stealing*. Retrieved from https://hackingstuffs.com/attacks/cookies-stealing/

The OWASP Foundation. (2017a). *The open web application security project*. Retrieved from https://www.owasp.org/index.php/CrossSite_Request_Forgery/

The OWASP Foundation. (2017b). *The open web application security project*. Retrieved from https://www.owasp.org/index.php/OWASP Periodic Table of Vulnerabilities_-_Insufficient_Transport_Layer_Protection/

Veracode. (2017). *Insecure cryptographic storage*. Retrieved from http://www.veracode.com/security/insecure-crypto/

A STUDY ON SECURITY FOR WEB-BASED APPLICATIONS

Nguyen Sy Hoa[a], Lai Thi Nhung[b], Dang Thanh Hai[c*]

[a]The Faculty of Information Technology and Telecommunication, VNU University of Science, Hanoi, Vietnam
[b]The Faculty of Basic Science, Namdinh University of Nursing, Namdinh, Vietnam
[c]The Faculty of Information Technology, Dalat University, Lamdong, Vietnam
[*]Corresponding author: Email: haidt@dlu.edu.vn

Article history
Received: March 10th, 2017 | Received in revised form: May 04th, 2017
Accepted: May 19th, 2017

Abstract

The Internet has been developing extremely rapidly, connecting more than 30% of the world population, with more than 2.2 billion users. It has brought benefits as well as risks. In 2005, the world lost over $445 billion while, Vietnam lost approximately 8,700 billion VNĐ due to the incidents of cyberattacks, in which 5,226 Websites of agencies and businesses in Vietnam were attacked. This is mainly due to the hackers' progress, the advent of new technologies, and the fact that today's systems are increasingly complex and it is difficult to manage all risks. Therefore, studying the risks for Web-based applications is the urgent need for organizations deploying web applications on the Internet. In this article, we will analyze the most common vulnerabilities of Web-based applications and recommend methods for detection.

Keywords: Security; Web-based applications; Web security.