Phát hiện điểm yếu phần mềm

Nguyễn Thành Nam, cisa, cissp, csslp

Ngày 18 tháng 12 năm 2010

Tóm tắt nội dung

Lỗi phần mềm có sự ảnh hưởng đặc biệt nghiêm trọng đối với mọi ngành nghề có sử dụng sự trợ giúp của máy tính như tài chính, quốc phòng, kỹ thuật. Việc phát hiện lỗi phần mềm, đặc biệt là lỗi an ninh, vì thế trở thành một công việc nhạy cảm, đòi hỏi nhiều sự sáng tạo, lẫn tính lý luận ở người thực hiện. Bài báo này trình bày một số quan điểm về công việc phát hiện lỗi an ninh trong bức tranh to hơn là công tác đảm bảo chất lượng sản phẩm. Bên cạnh đó, bài báo cũng đưa ra một số phương pháp chung đã được áp dụng nhiều trong kiểm thử phần mềm cổ điển để áp dụng vào việc phát hiện lỗi an ninh, cùng với một số ví dụ cụ thể, thực tế để miêu tả rõ bản chất vấn đề.

1 Vai trò của an ninh phần mềm

1.1 Vai trò của phần mềm

Phần mềm là bộ não của một hệ thống thông tin. Dù theo bất kỳ định nghĩa nào đi nữa thì một hệ thống thông tin luôn là tập hợp các công việc với mục đích xử lý thông tin [1]. Nếu không có phần mềm, những chiếc máy tính, hoặc thiết bị mạng không thể mang lại giá trị cho hệ thống thông tin vì chúng không thể hiểu thông tin đầu vào, hay tạo thông tin đầu ra. Phần mềm chính là yếu tố làm nên sự "thông minh" của hệ thống.

Hơn nữa, sự "thông minh" này xuất hiện khắp mọi nơi trong hệ thống thông tin. Phần mềm không chỉ là những ứng dụng riêng biệt trong ngành mà còn là hệ điều hành, là hệ quản lý cơ sở dữ liệu, là các ứng dụng máy chủ, là các phần dẻo (firmware) nhúng trong các phần cứng, là những vi lệnh của bộ vi xử lý, v.v... Nơi nào có trí tuệ để xử lý dữ liệu, nơi đó có phần mềm.

Điểm quan trọng nhất là phần mềm có ảnh hưởng trực tiếp đến tài sản của tổ chức, cá nhân sử dụng hệ thống thông tin. Vì phần mềm là trung tâm xử lý dữ liệu của hệ thống thông tin, phần mềm ảnh hưởng trực tiếp đến kết quả nhận được từ hệ thống. Kết quả cuối cùng này, cho dù đó là một sáng chế khoa học, hay một tiểu thuyết, thường là tài sản vô hình, có giá trị cao và mang ý nghĩa quan trọng đối với người sử dụng hệ thống. Do đó, tác động đến phần mềm sẽ gây ra các tác động trực tiếp đến tài sản.

1.2 Lỗi tính năng và lỗi an ninh

Vì lẽ đó nên sai sót (hay lỗi) phần mềm gây thiệt hại trực tiếp đến tài sản của tổ chức, cá nhân sử dụng hệ thống, đôi khi làm phá sản toàn bộ dự án [5]. Lỗi tính năng có thể làm dữ liệu biến nghĩa vì được xử lý sai. Nếu dữ liệu này được tiếp tục sử dụng trong các công đoạn sau nữa sẽ có thể tạo nên hiệu ứng đô-mi-nô, dẫn đến một chuỗi dây chuyền dữ liệu sai. Tuy nhiên, nhìn chung lỗi tính năng có phạm vi cục bộ, giới hạn trong ứng dụng cụ thể nào đấy, và có thể cả những ứng dụng phụ thuộc nó.

Ở đây, lỗi tính năng được mang ý nghĩa là những sai sót, lệch lạc so với yêu cầu kỹ thuật ban đầu, mang tính cơ bản của sản phẩm. Ví dụ như một chương trình máy tính cho ra kết quả 1,333*7390689020375894* khi thực hiện phép tính $\frac{4195835.0}{3145727.0}$, mà giá trị đúng phải là 1,333*8204491362410025* [15]. Trong khi đó, lỗi an ninh được hiểu là những lỗi có thể bị người khác trục lợi. Sự biểu hiện ra ngoài, hay khả năng tận dụng được của lỗi không quyết định một lỗi có phải là lỗi an ninh hay không. Ví dụ như một ứng dụng mạng cho phép người dùng đăng nhập mặc dù sai mật khẩu [4, 12].

Lỗi an ninh có ảnh hưởng rộng hơn lỗi tính năng. Lỗi tính năng gây thiệt hại cho người sử dụng trong khi lỗi an ninh không chỉ gây cùng thiệt hại mà còn đem lợi bất chính cho đối tượng khác. Lỗi an ninh xảy ra trong một ứng dụng có thể khiến toàn bộ hệ thống bị xâm nhập. Và nghiêm trọng hơn cả là lỗi an ninh, khi bị tận dụng, có thể không biểu hiện bất kỳ dấu hiệu nào ra bên ngoài để nhận biết. Những yếu tố trên khiến cho việc đánh giá thiệt hại tài sản trở nên khó khăn bội phần vì không rõ biểu hiện, không rõ điểm xuất phát lỗi, và không rõ những thành phần nào còn lỗi.

Nói tóm lại, lỗi phần mềm gây tổn thất lớn đến tài sản, cho dù là hữu hình như sự tê liệt thiết bị, hay vô hình như sự sợ hãi, bất an, và hoài nghi về hệ thống thông tin [17].

2 Công tác đảm bảo chất lượng phần mềm

2.1 Đảm bảo chất lượng

Công tác đảm bảo chất lượng phần mềm (Software Quality Assurance) là cách tiếp cận việc đánh giá chất lượng và sự tuân thủ theo chuẩn sản phẩm, hay quy trình một cách có kế hoạch và hệ thống [19]. SQA bao gồm các công tác xuyên suốt quá trình sản xuất phần mềm như thu thập yêu cầu, phân tích, thiết kế, phát triển, kiểm thử và triển khai. SQA xoay quanh hai công việc chính là xác minh (liệu tổng của 1 và 2 có phải là 3 hay không) và công nhận (liệu mục tiêu của chương trình có phải là tính tổng của 1 và 2 hay không) phần mềm nhằm cân-đo-đong-đếm hóa khái niệm "chất lượng".

"Chất lượng", xét cho cùng, phụ thuộc vào người sử dụng sản phẩm. Một ứng dụng chạy đúng nhưng cần nhiều thời gian chưa chắc đã thỏa mãn ý muốn của người dùng và do đó không đạt tiêu chuẩn của họ. Ngược lại, đôi khi ứng dụng đưa ra kết quả *gần* đúng trong khoảng thời gian chấp nhận được lại là một ứng dụng làm người dùng hài lòng, và do đó có chất lượng.

2.2 Tính nghệ thuật và khoa học

SQA là một hoạt động đòi hỏi tính khoa học. Công tác SQA tập trung vào việc đo lường những yếu tố cấu thành chất lượng sản phẩm. Để việc xác minh và công nhận chất lượng được khách quan, thì công việc đo lường các yếu tố này phải được thể hiện thành những con số minh bạch, từ các bước thực hiện rõ ràng, lập lại được, tương tự như những thí nghiệm khoa học.

Vì "chất lượng" là một khái niệm trừu tượng và phụ thuộc nhiều vào yếu tố chủ quan, nên công tác đảm bảo chất lượng cũng mang tính chủ quan, phụ thuộc nhiều vào sự sáng tạo, và tính nghệ thuật trong quá trình thực hiện. Những câu hỏi và trả lời cho các vấn đề như "tại sao chúng ta phải tập trung xác minh tính đúng đắn của hàm này hơn hàm kia" chính là sự đúc kết của kinh nghiệm làm việc.

Do đó, tương tự như chính công tác phát triển phần mềm [11], SQA vừa mang tính khoa học, vừa mang tính nghệ thuật.

Riêng về khía cạnh an ninh, việc phát hiện lỗi an ninh có xu hướng mang đậm tính nghệ thuật hơn kiểm thử tính năng. Các tính năng an ninh thường ít được quy định chặt chẽ trong đặc

điểm kỹ thuật của sản phẩm nên thường ít được nhớ đến khi thực hiện kiểm thử. Lý do thứ hai là các lỗi an ninh thường chỉ bị phát hiện trong những tình huống không lường trước. Sự nhạy bén, tinh tế trong việc lường trước những tình huống như thế này chính là tính nghệ thuật, sự sáng tạo xuất phát từ kinh nghiệm và không thể được đào tạo rập khuôn [18].

3 Phát hiện lỗi an ninh

Mặc dù mang đậm tính nghệ thuật nhưng kiểm thử an ninh cũng có thể áp dụng một số phương pháp khoa học nhằm "hái trái thấp", lược bớt những lỗi an ninh dễ mắc phải.

3.1 Phương pháp chung

Một số phương pháp phát hiện lỗi tính năng đã được nghiên cứu nhiều trong công tác SQA [10, 2, 9] có thể được sử dụng một cách hiệu quả để phát hiện lỗi an ninh. Các phương pháp này áp dụng tốt nhất trong quá trình kiểm thử hộp trắng.

3.1.1 Dựa vào kiểu dữ liệu

```
char s1[8];
char s2[8];
strcpy(s2, "");
strcpy(s1, "12345678");
strcat(s2, s1);
```

Ví dụ 1: Kết thúc chuỗi

```
void transfer(Account receiver, int amount)
{
  this.balance -= amount;
  receiver.balance += amount;
}
```

Ví dụ 2: Sai về dấu

Kiểu chuỗi cần chú ý đến sự tồn tại của ký tự kết thúc chuỗi (trong ngôn ngữ C), xem xét nếu chuỗi rỗng, hay dài quá mức, hoặc dài vừa đúng. Trong ví dụ 1, sau khi thực hiện hàm `strcpy` thì chuỗi `s1` không còn ký tự kết thúc chuỗi, và khi được chép qua `s2` sẽ có thể gây lỗi tràn bộ đệm.

Kiểu số nguyên cần quan tâm đến các vấn đề về dấu (âm hay dương), về độ lớn của kiểu dữ liệu (1 byte, 2 byte, 4 byte, hay 8 byte), về giá trị ngưỡng. Trong ví dụ 2, vì biến `amount` là kiểu số nguyên có dấu nên nó có thể là một số âm và sẽ khiến cho việc *chuyển tiền* từ một tài khoản thành việc *thêm tiền* cho tài khoản đó.

3.1.2 Dựa vào giá trị dữ liệu

```
char value[512];
RegSetValue(..., 512);
...
RegQueryValue(..., 4096);
```

Ví dụ 3: Trích từ BKAV của BKAV

```
int pids[128]; int i = 0;
while (HasNextProcess()) {
  pids[i++] = NextProcess().Id;
}
```

Ví dụ 4: Trích từ BKAV của BKAV

Giá trị ngưỡng (cả ngưỡng trên lẫn ngưỡng dưới) có thể được kiểm tra bằng cách sử dụng các giá trị dữ liệu vừa trên ngưỡng, ngay đúng ngưỡng, vừa dưới ngưỡng, và chú ý đến

các phép so sánh có bằng (lớn hơn hoặc bằng, bằng, nhỏ hơn hoặc bằng). Trong ví dụ 3, mảng `value` được xác định chứa 512 ký tự và được sử dụng đúng khi thiết lập giá trị registry (ở hàm `RegSetValue`) nhưng khi lấy giá trị registry thì khai báo quá lớn (ở hàm `RegQueryValue`), gây tràn bộ đệm [3].

Vòng lặp có thể được kiểm tra tương tự như giá trị ngưỡng: xem số lần lặp (n) là một giá trị ngưỡng và kiểm tra việc việc lặp $n+1$ vòng, n vòng, $n-1$ vòng, 1 vòng, hoặc bỏ hẳn vòng lặp. Trong ví dụ 4, lập trình viên đã giả định chỉ có tối đa 128 tiến trình trên hệ thống và không kiểm tra giả định đó trong quá trình chạy [3].

3.1.3 Dựa vào hoàn cảnh sử dụng

```
char filename[...];
char *cid;
snprintf(filename, ..., "attachment\\%s", cid);
```

Ví dụ 5: Trích từ eOffice của BKAV

Một số dữ liệu do người dùng cung cấp được sử dụng để tạo nên các dữ liệu đầu vào cho các hàm, các thành phần liên quan của chương trình. Trong các trường hợp đó, hoàn cảnh sử dụng phải được xem xét với các câu hỏi liên quan đến định dạng dữ liệu, cấu trúc dữ liệu, giá trị phù hợp (mặc định là cấm, hay mặc định là cho phép). Trong ví dụ 5, `cid` là một dữ liệu do người dùng cung cấp, và nó được sử dụng để tạo tên tập tin. Nếu `cid` có giá trị bắt đầu bằng `..\` thì sẽ khiến cho tên tập tin vượt ra ngoài thư mục `attachment` đã được chỉ định.

Lỗi liên quan đến ngữ cảnh sử dụng thường được đánh giá là loại lỗi nguy hiểm nhất. Trong cả hai bảng xếp hạng năm 2010 của CWE/SANS [21] và OWASP [16], loại lỗi này đều chiếm hai thứ hạng đầu (lỗi kịch bản chéo trang, và lỗi chèn truy vấn). Chúng được xem là những cú móc trái, phải làm đo ván đa số các ứng dụng. Lỗi kịch bản chéo trang liên quan đến định dạng dữ liệu *xuất* và lỗi chèn truy vấn liên quan đến giá trị phù hợp của dữ liệu *nhập*.

```
<?php
echo 'Hello ' .
    $_GET['name'];
?>
```

Ví dụ 6: Kịch bản chéo trang

```
String empId, query;
empId = req.getParameter("empId");
query = "SELECT * FROM Account WHERE id = '"
    + empId + "'";
```

Ví dụ 7: Chèn truy vấn

Trong ví dụ 6, nếu tham số `name` có chứa những ký tự đặc biệt, ví dụ như các ký tự bắt đầu và kết thúc thẻ HTML, thì dữ liệu xuất ra sẽ có ý nghĩa riêng đối với trình duyệt. Trong ví dụ 7, nếu `empId` có giá trị là `'or'1'='1` thì câu truy vấn trở thành **SELECT** * **FROM** Account **WHERE** id = ''**or**'1'='1' và sẽ làm thay đổi kết quả truy vấn.

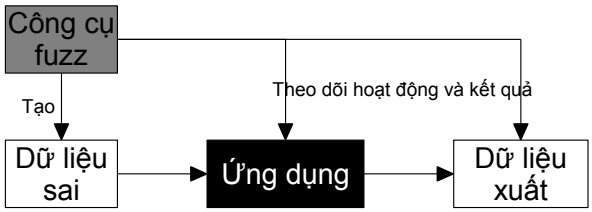
3.1.4 Dựa vào các hàm nguy hiểm

Một số hàm trong thư viện chuẩn thiếu phần kiểm tra độ lớn dữ liệu (ví dụ như `strcpy`, `strcat`, `sprintf`, `vsprintf`), có những hiệu ứng phụ ít được để ý tới (ví dụ `strtok`, `snprintf`), hoặc rất khó để được sử dụng đúng cách (ví dụ `strncpy`, `strncat`). Việc rà soát sự xuất hiện của các hàm này [8] trong mã nguồn phần mềm là một cách đơn giản để hạn chế lỗi an ninh.

Ngoài ra, rà soát việc sử dụng của các hàm quan trọng như các hàm suất dữ liệu, các hàm thực hiện truy vấn dữ liệu cũng là một cách hữu hiệu trong việc phát hiện các lỗi liên quan tới hoàn cảnh sử dụng. Trong ví dụ 7, chúng ta có thể tìm thấy lỗi bằng cách tìm hàm `executeQuery`

và lần ngược lại xem dữ liệu nhập vào hàm này là gì. Khi đó ta sẽ tìm tới biến `query` và phát hiện rằng dữ liệu nhập `empId` chưa được chuyển đổi phù hợp.

3.2 Phương pháp mờ (Fuzzing)



Hình 1: Phương pháp mờ

```
what = packet[:]
where = choice(range(len(packet)))
which = chr(choice(range(256)))
what[where] = which
```

Ví dụ 8: Kiểm tra mờ SMBv2

Phương pháp mờ [20] (hay còn được biết đến như kỹ thuật kiểm tra sinh lỗi [2]) hoạt động bằng cách tạo ra những mẫu dữ liệu sai lệch, không lường trước, hoặc dữ liệu ngẫu nhiên và sử dụng chúng trong chương trình. Khi chương trình hoạt động sai, hoặc sinh ra lỗi thì lỗi này sẽ được ghi nhận lại (xem hình 1). Do ít quan tâm đến phương thức xử lý dữ liệu của chương trình nên phương pháp mờ có thể được sử dụng trong các kiểm thử hộp đen.

Phương pháp này bắt đầu được sử dụng ngày càng nhiều hơn trong những năm gần đây vì chúng có thể được tự động hóa, triển khai hàng loạt (trên mô hình phân bố với các máy tính mạng [7], hoặc trên một máy tính với các máy ảo [14]), và hoạt động rất nhanh. Phương pháp mờ giúp giảm bớt sự phụ thuộc vào người kiểm tra nhưng làm tăng tính chất may rủi của việc phát hiện lỗi. Ví dụ 8 là bốn dòng quan trọng nhất của một đoạn mã thực hiện việc kiểm tra giao thức Server Message Block phiên bản 2 [6]. Mặc dù rất đơn giản nhưng đoạn mã này đã phát hiện một lỗi nghiêm trọng khiến Windows 7 bị kiểm soát hoàn toàn [13]. Điều đáng nói là chương trình chỉ mất ba giây để phát hiện ra lỗi.

4 Tóm tắt

Phần mềm đóng vai trò "bộ não" của một hệ thống thông tin và có ảnh hưởng đến dữ liệu, một loại tài sản vô hình, của tổ chức hoặc cá nhân sử dụng hệ thống đó. Lỗi phần mềm gây ra thiệt hại trực tiếp đến tài sản này và rất khó để phát hiện chúng, đặc biệt là những lỗi an ninh.

Công tác đảm bảo chất lượng phần mềm tập hợp những phương pháp giúp phát hiện những điểm yếu của sản phẩm phần mềm. Công tác này do đó đòi hỏi sự đo lường minh bạch, rõ ràng, mang tính khoa học, đồng thời cũng là sự đúc kết kinh nghiệm, sự nhạy bén, và sáng tạo nghệ thuật của người thực hiện.

Mặc dù mang tính nghệ thuật cao hơn nhưng việc phát hiện lỗi an ninh cũng có thể sử dụng một số phương pháp kiểm thử chung để lược đi những lỗi cơ sở. Ngoài ra, sự xuất hiện và phát triển của những công cụ, tài liệu liên quan đến phương pháp mờ cũng góp phần thúc đẩy việc áp dụng nó trong quá trình phát hiện lỗi trong những năm gần đây.

Tham khảo

[1] Alter, S. *The Work System Method: Connecting People, Processes, and IT for Business Results*. Work System Press, 2006.

[2] Beizer, B. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[3] Blue Moon Consulting. Multiple buffer overflows in bkav pro. http://www.bluemoon.com.vn/advisories/bmsa200806.html, June 2008.

[4] Blue Moon Consulting. Authentication bypass in interspire shopping cart. http://www.bluemoon.com.vn/advisories/bmsa200901.html, January 2009.

[5] Dowson, M. The ariane 5 software failure. *SIGSOFT Softw. Eng. Notes 22* (March 1997), 84–.

[6] Gaffié, L. More explication on cve-2009-3103. http://g-laurent.blogspot.com/2009/10/more-explication-on-cve-2009-3103.html, October 2009.

[7] Gallagher, T., and Conger, D. Office security engineering. Microsoft. BlueHat09.

[8] Howard, M., and Lipner, S. *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA, 2006.

[9] Jorgensen, P. C. *Software Testing: A Craftsman's Approach*, 3rd ed. Auerbach Publications, Feb 2008.

[10] Kaner, C., Falk, J. L., and Nguyen, H. Q. *Testing Computer Software, Second Edition*, 2nd ed. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[11] McConnell, S. The art, science, and engineering of software development. *IEEE Softw. 15* (January 1998), 120–119.

[12] Microsoft. Microsoft security bulletin (ms00-072). http://www.microsoft.com/technet/security/bulletin/ms00-072.mspx, October 2000.

[13] Microsoft. Microsoft security bulletin ms09-050 – critical. http://www.microsoft.com/technet/security/bulletin/ms09-050.mspx, October 2009.

[14] Nagy, B. Industrial bug mining - extracting, grading and enriching the ore of exploits. COSEINC. Black Hat USA 2010.

[15] Nicely, T. R. Pentium fdiv flaw. http://www.trnicely.net/pentbug/pentbug.html, August 2010.

[16] OWASP. Owasp top 10 application security risks – 2010. http://www.owasp.org/index.php/Top_10_2010-Main, April 2010.

[17] Schneier, B. *Beyond Fear: Thinking Sensibly about Security in an Uncertain World*. Copernicus Books, 2003.

[18] Schneier, B. The ethics of vulnerability research. http://www.schneier.com/crypto-gram-0805.html, May 2008. Crypto-Gram, bản dịch tại http://www.bluemoon.com.vn/articles/the.ethics.of.vulnerability.research.html.

[19] Software Assurance Technology Center–NASA. Software assurance guidebook. http://satc.gsfc.nasa.gov/assure/agb.txt.

[20] Takanen, A., DeMott, J., and Miller, C. *Fuzzing for Software Security Testing and Quality Assurance*, 1st ed. Artech House, Inc., Norwood, MA, USA, 2008.

[21] The MITRE Corporation. 2010 cwe/sans top 25 most dangerous software errors. http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.pdf, December 2010.