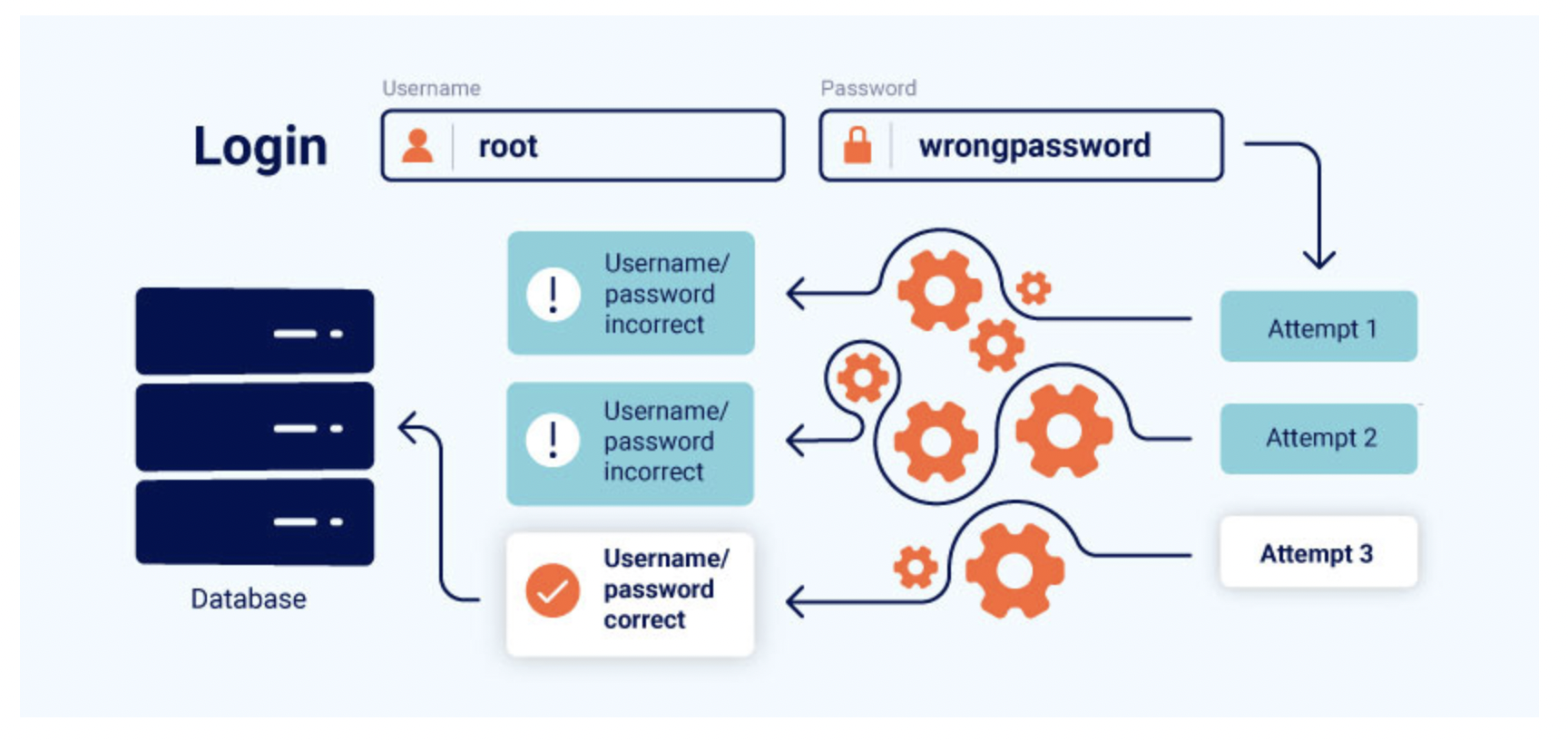
**Business logic vulnerabilities**



**Khái niệm**

Lỗ hổng logic business là các lỗi logic và thực thi của 1 ứng dụng cho phép kẻ tấn công gợi ra các hành vi không mong muốn. Điều này cho phép kẻ tấn công lạm dụng chức năng hợp pháp để đặt được mục đích xấu. Những sai sót này thường là kết quả của việc không lường trước được các trạng thái bất thường của ứng dụng có thể xảy ra và do đó không xử lí chúng 1 một cách an toàn.

Lưu ý: trong bối cảnh này, thuật ngữ “business logic” nhắc đến 1 bộ các quy tắc cách ứng dụng hoạt động. Vì các quy tắc này không liên quan trực tiếp đến business, các lỗ hổng liên quan cũng được biết đến là lỗ hổng logic của ứng dụng hay “logic flaws”.

Lỗi logic thường không nhìn thấy được vì nó không được hiển thị trên ứng dụng. Tuy nhiên, kẻ tấn công có thể khai thác hành vi của ứng dụng bằng cách tương tác với nó theo nhiều cách khác nhau mà nhà phát triển không thể lường trước được.

Một trong những mục đích chính của business logic là thực thi các quy tắc và ràng buộc được áp dụng khi thiết kế ứng dụng hay chức năng của nó. Nhìn chung, các quy tắc business ra lệnh cho ứng dụng xử lí như thế nào khi 1 diễn cảnh xảy ra. Điều này bao gồm việc ngăn chặn người dùng không tác động xấu đến ứng dụng.

Các lỗi logic có thể cho phép kẻ tấn công phá vỡ các quy tắc ấy. Ví dụ, họ có thể hoàn thành 1 giao dịch mà không cần trải qua 1 quy trình mua hàng dự kiến. Nói cách khác, việc xác thực dữ liệu do người dùng cung cấp bị phá vỡ cho phép kẻ tấn công tạo ra những thay đổi lớn đến giá trị giao dịch hay gửi 1 đầu vào vô nghĩa. Bằng cách gửi các giá trị không mong muốn cho logic ở phía máy chủ, kẻ tấn công có thể lôi kéo ứng dụng làm 1 việc gì đó mà nó không ngờ tới.

Các lỗ hổng logic thường rất phong phú và độc nhất với từng ứng dụng và chức năng của nó. Việc xác định chúng yêu cầu 1 lượng kiến thức nhất định như hiểu về business domain hay phần thưởng kẻ tấn công có thể giành được trong bối cảnh đó. Điều này làm cho chúng rất khó bị phát hiện bởi các công cụ quét lỗ hổng tự động. Do đó, lỗi logic là 1 mục tiêu đáng nhắm tới của các thợ săn tiền thưởng và các tester thủ công nói chung.

**Lỗ hổng business logic sinh ra như thế nào?**

Các lỗ hổng logic thường sinh ra vì đội thiết kế và phát triển thường đưa ra những giả định sai lầm về cách người dùng sẽ tương tác với ứng dụng. Những giả định tệ đó có thể dẫn đến việc xác nhận đầu vào của người dùng không đầy đủ. Ví dụ, nếu nhà phát triển giả định người dùng chỉ gửi dữ liệu qua trình duyệt web, ứng dụng có thể hoàn toàn dựa vào sự kiểm soát ở phía máy khách để xác thực. Điều này rất dễ bị bỏ qua bởi kẻ tấn công bằng intercepting proxy.

Cuối cùng, điều này đồng nghĩa với việc khi 1 kẻ tấn công đi chệch hướng so với hành vi mong muốn, ứng dụng sẽ không thể ngăn chặn điều đó và xử lí không an toàn.

Lỗi logic thường rất phổ biến trong các hệ thống phức tạp mà ngay cả nhóm phát triển cũng không thể hiểu hết. Để tránh lỗi logic, các nhà phát triển cần hiểu toàn bộ ứng dụng, bao gồm việc hiểu các chức năng khác nhau có thể được kết hợp theo các cách không mong muốn. Các nhà phát triển làm việc trên 1 lượng code lớn có thể không hiểu sâu sắc cách tất cả khu vực của ứng dụng hoạt động. Một vài người làm việc trên 1 component có thể đưa ra những giả định sai về cách 1 component khác hoạt động, do đó sinh ra lỗ hổng logic nghiêm trọng. Nếu nhà phát triển không ghi lại rõ ràng bất cứ giả định nào đã xảy ra, những loại lỗ hổng này rất dễ xâm nhập vào ứng dụng.

**Ảnh hưởng của lỗ hổng business logic**

Ảnh hưởng của lỗ hổng business logic đôi khi rất tầm thường. Đó là 1 phạm trù rộng và tác động của nó thường rất khác nhau. Tuy nhiên, bất cứ hành vi không mong muốn nào cũng có thể dẫn đến các cuộc tấn công có mức độ nghiêm trọng cao nếu kẻ tấn công có thể tận dụng ứng dụng đúng cách. Với lí do đó, các lỗi logic kì quặc cũng nên được fixed kể cả khi bạn không tìm thấy cách khai thác chúng. Nó luôn có rủi ro khi một ai đó sẽ làm được.

Nhìn chung, ảnh hưởng của lỗi logic dựa vào chức năng mà nó liên quan. Nếu lỗi nằm ở cơ chế xác thực, nó có thể ảnh hưởng rất lớn đến toàn bộ lớp bảo mật của bạn. Kẻ tấn công có thể khai thác nó để leo thang đặc quyền, hoặc bỏ qua toàn bộ xác thực, giành quyền truy cập dữ liệu và chức năng nhạy cảm. Điều này cũng gia tăng phạm vi tấn công cho những khai thác khác.

Lỗi logic trong giao dịch tài chính rõ ràng dẫn đến sự mất mát lớn về kinh tế.

Bạn cũng nên lưu ý mặc dù lỗi logic không cho kẻ tấn công lợi ích trực tiếp nhưng chúng cho phép bên thứ ba gây thiệt hại về kinh tế theo 1 cách nào đó.

**Khai thác lỗ hổng Business Logic**

**Tin tưởng quá mức vào các biện pháp kiểm soát ở phía máy khách**

Một giả định sai lầm là người dùng chỉ có thể tương tác với ứng dụng qua giao diện của trang web dẫn đến 1 giả định khác là xác thực ở phía máy khách sẽ ngăn chặn người dùng cung cấp đầu vào độc hại. Tuy nhiên, kẻ tấn công có thể dùng công cụ Burp Proxy để giả mạo dữ liệu sau khi nó được gửi đi bởi trình duyệt và trước khi được xử lí bởi logic của máy chủ. Điều này làm cho các biện pháp kiểm soát ở phía máy khách trở nên vô nghĩa.

Chấp nhận dữ liệu mà không kiểm tra tính toàn vẹn, xác thực ở phía máy chủ có thể cho phép kẻ tấn công gây ra mọi loại thiệt hại với rất ít nỗ lực. Cái họ có thể nhận được dựa vào chức năng và chức năng đó làm gì với dữ liệu có thể kiểm soát được. Loại lỗi này có thể có những hậu quả rất tàn khốc cho cả chức năng liên quan đến business và bảo mật của trang web.

**Không xử lí được đầu vào bất thường**

Một mục đích của logic ứng dụng là hạn chế đầu vào của người dùng vi phạm các qui tắc liên quan đến business. Ví dụ, ứng dụng có thể được thiết kế để chấp nhận giá trị bất kỳ nhưng logic của ứng dụng sẽ quyết định giá trị đó phụ thuộc vào business. Rất nhiều ứng dụng kết hợp numeric limits trong logic. Nó được thiết kế để quản lí hàng tồn kho, giới hạn ngân sách,…

Cùng lấy 1 ví dụ về cửa hàng online. Khi đặt sản phẩm, người dùng chỉ định số lượng họ muốn đặt. Mặc dù về lí thuyết integer là đầu vào hợp lệ nhưng business logic có thể ngăn chặn người dùng đặt hàng nhiều hơn số lượng hiện có trong kho.

Để thực thi được những qui tắc như vậy, các nhà phát triển cần đoán trước tất cả diễn cảnh và các cách kết hợp để xử lí chúng trong logic của ứng dụng. Nói cách khác, họ cần nói cho ứng dụng biết khi nào nó nên chấp nhận đầu vào đó và nên xử lí dựa vào nhiều điều kiện khác nhau. Nếu không có logic rõ ràng để xử lí những trường hợp cụ thể, điều đó có thể dẫn đến các hành vi không mong muốn và có thể khai thác được.

Ví dụ, kiểu dữ liệu số có thể chấp nhận số âm. Nếu ứng dụng không xác thực rõ ràng ở phía máy chủ, kẻ tấn công có thể gửi giá trị âm và gây ra hành vi không mong muốn.

Xem xét việc chuyển tiền giữa hai tài khoản ngân hàng. Chức năng này hầu hết sẽ kiểm tra người gửi có đủ tiền trước khi thực hiện giao dịch:

$transferAmount = $\_POST['amount'];

$currentBalance = $user->getBalance();

if ($transferAmount <= $currentBalance) {

// Complete the transfer

} else {

// Block the transfer: insufficient funds

}

Nếu logic không ngăn chặn người dùng cung cấp giá trị âm trong tham số amount, kẻ tấn công có thể bỏ qua balance check và chuyển tiền sai hướng. Nếu họ gửi -1000$ đến tài khoản nạn nhân, họ sẽ nhận được $1000 từ nạn nhân đó.

Một lỗi logic như vậy có thể rất nguy hiểm nếu chúng xuất hiện trong đúng chức năng. Chúng cũng rất dễ bị bỏ qua trong quá trình phát triển và kiểm thử, đặc biệt những đầu vào như vậy có thể bị chặn ở phía máy khách.

Khi kiểm thử 1 ứng dụng, bạn nên dùng các công cụ như Burp Proxy và Repeater để thử gửi các giá trị không mong muốn. Đặc biệt, thử nhập phạm vi mà người dùng hợp lệ khó có thể nhập vào. Điều này có thể bao gồm các trường hợp ngoại lệ như 1 số rất nhỏ hay 1 chuỗi rất dài. Bạn có thể thử nhiều kiểu dữ liệu khác nhau. Bằng việc quan sát phản hồi của ứng dụng, bạn nên trả lời những câu hỏi sau:

* Có bất cứ limits nào được hiển thị trên dữ liệu?
* Chuyện gì sẽ xảy ra khi bạn chạm đến limits đó?
* Có bất kì sự chuyển đổi hay chuẩn hóa nào được thực hiện trên dữ liệu của bạn?

Điều này có thể làm lộ ra khả năng xác thực đầu vào yếu, cho phép bạn tận dụng ứng dụng theo nhiều cách khác nhau. Hãy nhớ rằng nếu bạn tìm thấy 1 trang web không xử lí đầu vào khác thường một cách an toàn thì có khả năng những chức năng khác cũng gặp vấn đề tương tự.

**Đưa ra những giả định sai về hành vi của người dùng**

**Người dùng đáng tin cậy không phải lúc nào cũng đáng tin cậy**

Các ứng dụng trông có vẻ bảo mật vì chúng thực hiện những biện pháp nghiêm ngặt để thực thi các quy tắc business. Không may mắn, một vài ứng dụng đã đưa ra giả định sai rằng vượt qua các biện pháp kiểm soát nghiêm ngặt ban đầu, người dùng và dữ liệu của họ có thể được tin tưởng vô thời hạn. Điều này có thể dẫn đến việc thực thi tương đối lỏng lẻo các biện pháp kiểm soát tương tự kể từ thời điểm đó trở đi.

Nếu các quy tắc business và phương pháp bảo mật không được áp dụng đồng bộ trong suốt ứng dụng, điều này có thể dẫn đến những lỗ hổng tiềm ẩn nguy hiểm có thể được khai thác bởi kẻ tấn công.

**Người dùng không phải lúc nào cũng cung cấp thông tin bắt buộc**

Một giả định sai là người dùng sẽ luôn cung cấp giá trị cho các trường nhập bắt buộc. Các trình duyệt có thể ngăn chặn người dùng thông thường gửi biểu mẫu mà không nhập thông tin bắt buộc, nhưng như chúng ta đã biết, kẻ tấn công có thể giả mạo tham số trong quá trình truyền đi, thậm chí là loại bỏ hoàn toàn tham số.

Đây là vấn đề cụ thể trong các trường hợp nhiều hàm được thực thi trong cùng 1 tập lệnh ở phía máy chủ. Trong trường hơp đó, sự hiện diện hay vắng mặt của 1 tham số cụ thể có thể quyết định code nào được thực thi. Loại bỏ các giá trị của tham số cho phép kẻ tấn công truy cập vào các đường dẫn code nằm ngoài tầm với.

Khi thăm dò lỗi logic, bạn nên thử loại bỏ mỗi tham số và quan sát hành vi của ứng dụng. Bạn nên đảm bảo là:

* Chỉ loại bỏ 1 tham số mỗi lần để đảm bảo đạt được tất cả các đường dẫn code liên quan
* Thử xóa tên của tham số và giá trị. Máy chủ sẽ xử lí hai trường hợp này rất khó khăn.
* Theo dõi các quá trình nhiều bước đến khi hoàn thành. Thỉnh thoảng giả mạo 1 tham số ở 1 bước sẽ có ảnh hưởng đến 1 bước khác trong quá trình làm việc.

Điều này áp dụng cho cả Url và tham số Post, nhưng đừng quên kiểm tra cookies nữa. Quá trình này có thể tiết lộ một vài hành vi kì lạ của ứng dụng mà có thể khai thác được.

**Người dùng không phải lúc nào cũng làm theo trình tự đã định**

Rất nhiều giao dịch thanh toán có nhiều bước. Giao diện web sẽ hướng dẫn người dùng thực hiện quá trình đó, đưa họ qua bước tiếp theo khi họ hoàn thành bước hiện tại. Tuy nhiên, kẻ tấn công không nhất thiết phải tuân theo trình tự đã định. Không tính đến khả năng này có thể dẫn đến lỗi logic có thể khai thác được.

Ví dụ, rất nhiều trang web thực thi xác thực hai bước yêu cầu người dùng đăng nhập 1 trang trước khi nhập mã xác thực ở 1 trang khác. Giả sử người dùng sẽ luôn tuân theo quy trình này cho đến khi hoàn thành và do đó, việc không xác minh rằng họ có làm như vậy hay không có thể cho phép kẻ tấn công bỏ qua hoàn toàn 2FA.

Với công cụ như Burp Proxy và Repeater, khi kẻ tấn công nhận được yêu cầu, họ có thể lặp lại nó và thực hiện mọi tương tác với máy chủ theo trình tự họ muốn. Điều này cho phép họ hoàn thành các hành động khác nhau trong khi ứng dụng đang ở trạng thái không mong muốn.

Để xác định lỗi logic này, bạn nên gửi yêu cầu theo trình tự không mong muốn. Ví dụ, bạn có thể bỏ qua 1 số bước, truy cập 1 bước nhiều hơn 1 lần, quay về các bước trước. Ghi chú lại các bước khác nhau được truy cập như thế nào. Mặc dù bạn hay gửi yêu cầu bằng Get hay Post đến 1 url chỉ định, thỉnh thoảng bạn có thể truy cập các bước bằng việc gửi nhiều bộ tham số khác nhau đến cùng 1 url. Với tất cả lỗi logic như vậy, cố gắng xác định nhà phát triển đã đưa ra những giả định nào và phạm vi tấn công nằm ở đâu. Sau đó, bạn có thể tìm cách để vi phạm những quy tắc đó.

Lưu ý là loại kiểm thử này thường sẽ gây ra các ngoại lệ vì các biến mong muốn có giá trị null hoặc không được khởi tạo. Chú ý đến bất cứ thông báo lỗi hay debug nào mà bạn gặp phải. Nó có thể dính lỗ hổng rò rỉ thông tin, có thể giúp bạn hiểu được rõ hơn về hành vi của ứng dụng.

**Lỗi domain-specific**

Trong rất nhiều trường hợp, bạn sẽ gặp phải lỗi logic chỉ dành riêng cho business domain hay mục đích của trang web.

Chức năng giảm giá của cửa hàng online là phạm vi tấn công quen thuộc khi săn lỗi logic. Nó có thể là 1 mỏ vàng tiềm năng cho kẻ tấn công với tất cả lỗi logic cơ bản có thể xảy ra theo cách mã giảm giá được áp dụng.

Ví dụ, xem xét 1 cửa hàng online cho mã giảm giá 10% cho đơn đặt hàng trên 1000$. Điều này có thể dễ bị lạm dụng nếu business logic thất bại trong việc kiểm tra đơn đặt hàng có thay đổi sau khi mã giảm giá được áp dụng không. Trong trường hợp này, kẻ tấn công có thể mua hàng chạm đến ngưỡng 1000$, sau đó họ xóa những hàng mình không muốn trước khi đặt hàng. Họ sẽ được giảm giá trên đơn đặt hàng mặc dù nó không vượt ngưỡng 1000$.

Bạn nên chú ý bất kỳ trường hợp nào mà giá hay những giá trị nhạy cảm khác được điều chỉnh dựa trên tiêu chí được quyết định bởi hành động của người dùng. Cố gắng hiểu thuật toán mà ứng dụng dùng để đưa ra những sự điều chỉnh và điểm nào được điều chỉnh. Nhờ đó ta có thể thao túng được ứng dụng khi sự điều chỉnh không tương ứng với chính sách ban đầu được dự định bởi nhà phát triển.

Để xác định những lỗ hổng đó, bạn cần nghĩ xem kẻ tấn công có những mục tiêu nào và cố gắng tìm nhiều cách khác nhau để làm được điều đó với chức năng có sẵn. Điều này có thể yêu cầu 1 mức độ kiến thức nhất định về domain-specific để hiểu điều gì có thể có lợi ở trong 1 bối cảnh nhất định. Ví dụ, bạn cần hiểu mạng xã hội để hiểu được những lợi ích của việc buộc một lượng lớn người dùng theo dõi bạn.

Không có kiển thức về domain, bạn có thể bỏ qua những hành vi nguy hiểm vì bạn không nhận thức được những ảnh hưởng tiềm tàng của nó. Tương tự như vậy, bạn có thể gặp khó khăn khi nối các điểm lại với nhau và để ý cách hai chức năng có thể được kết hợp theo 1 cách độc hại. Để đơn giản , các ví dụ được dùng trong chủ đề này liên quan đến 1 domain mà tất cả người dùng đã quen thuộc với nó là cửa hàng online. Tuy nhiên, khi bạn là thợ săn tiền thưởng, nhà kiểm thử, thậm chí là lập trình viên cố gắng viết mã bảo mật hơn, bạn cũng có thể gặp phải những domain ít gần gũi. Trong trường hợp đó, bạn nên đọc nhiều tài liệu nhất có thể và nếu có cơ hội, hãy nói chuyện với những chuyên gia về domain đó để có cái nhìn sâu hơn. Điều này nghe có vẻ mất rất nhiều thời gian, nhưng domain càng khó hiểu, các kiểm thử viên khả năng cao sẽ bỏ qua rất nhiều lỗi.

**Cung cấp 1 oracle mã hóa**

Diễn cảnh nguy hiểm có thể xảy ra khi đầu vào do người dùng kiểm soát được mật mã hóa và bản mã thu được sau đó sẽ cung cấp cho người dùng theo 1 cách nào đó. Loại đầu vào này thường được biết đến là “encryption oracle”. Kẻ tấn công có thể dùng đầu vào này để mã hóa dữ liệu bất kì bằng cách dùng đúng thuật toán và secret key.

Điều này trở nên nguy hiểm khi những đầu vào khác do người dùng kiểm soát cũng nhận dữ liệu được mã hóa bằng thuật toán tương tự. Trong trường hợp đó, kẻ tấn công có thể dùng encryption oracle để tạo 1 đầu vào được mã hóa hợp lệ để gửi nó vào những chức năng nhạy cảm.

Vấn đề này có thể phức tạp nếu đầu vào khác do người dùng kiểm soát cung cấp hàm truy ngược. Điều này cho phép kẻ tấn công giải mã dữ liệu khác để xác định được cấu trúc mong muốn. Nó giúp họ tiết kiệm rất nhiều thời gian để tạo 1 dữ liệu độc hại nhưng không nhất thiết phải thực hiện khai thác thành công.

Mức độ nghiêm trọng của encryption oracle phụ thuộc vào chức năng sử dụng chung thuật toán với oracle.