HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

HỌC PHẦN: KỸ THUẬT THEO DÕI VÀ GIÁM SÁT AN TOÀN MẠNG

BÁO CÁO BÀI TẬP CUỐI KỲ

Giảng viên:	**Ths.Ninh Thị Thu Trang**
Nhóm môn học**:**	02
Nhóm sinh viên thực hiện**:**	
Trịnh Thành Long	B18DCAT151
Ma Công Thành	B18DCAT235
Vũ Thị Huệ	B18DCAT103
Hoàng Đức Thắng	B18DCAT239

Hà Nội, 2022

Mục Lục

Lời mở đầu

Ngày nay, bạn cần kiểm soát hoàn toàn các sự cố mạng tiềm ẩn, đặc biệt là khi nói đến an ninh. Ngoài ra, có một cái nhìn tổng quan toàn cầu về chúng: nguyên nhân, tác động đến các nhiệm vụ hàng ngày và các giải pháp có thể được áp dụng. Thời gian đang chạy, buộc kết nối phải đáng tin cậy và cung cấp bảo vệ chống lại nhiều mối đe dọa.

Mạng bảo mật và quản lý mạng đang phát triển một cách nhanh chóng nhờ các công cụ giúp mọi thứ trở nên dễ dàng và thiết thực hơn nhiều. Đã qua rồi cái thời mà nhiều giải pháp, rất đắt và khó sử dụng, không đưa ra câu trả lời mong muốn. Các cuộc tấn công mạng đang ngày càng ít đình chiến hơn và các mạng phải có một lá chắn bảo vệ thực sự.

Zeek được trình bày như một công cụ để hỗ trợ quản lý ứng phó sự cố an ninh. Nó hoạt động bằng cách bổ sung dựa trên chữ ký các công cụ để tìm và theo dõi các sự kiện mạng phức tạp. Nó được đặc trưng bằng cách cung cấp phản hồi nhanh, ngoài việc sử dụng nhiều luồng và giao thức. Nó không chỉ giúp xác định các sự kiện bảo mật, mà còn nhằm mục đích tạo điều kiện khắc phục sự cố.

Nhận xét của giảng viên

....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....
....

Ths.Ninh Thị Thu Trang

I. Zeek là gì?

Zeek là một công cụ phân tích lưu lượng mạng mã nguồn mở thụ động. Nhiều nhà khai thác sử dụng Zeek như một trình giám sát an ninh mạng (NSM) để hỗ trợ điều tra hoạt động đáng ngờ hoặc độc hại. Zeek cũng hỗ trợ một loạt các nhiệm vụ phân tích lưu lượng ngoài miền bảo mật, bao gồm đo lường hiệu suất và khắc phục sự cố.

Lợi ích đầu tiên mà người dùng mới thu được từ Zeek là tập hợp các bản ghi mở rộng mô tả hoạt động mạng. Các bản ghi này không chỉ bao gồm bản ghi toàn diện của mọi kết nối được nhìn thấy trên dây mà còn bao gồm bản ghi lớp ứng dụng. Chúng bao gồm tất cả các phiên HTTP với URI được yêu cầu, tiêu đề khóa, loại MIME và phản hồi của máy chủ; Yêu cầu DNS với các câu trả lời; Chứng chỉ SSL; nội dung chính của các phiên SMTP; và nhiều hơn nữa. Theo mặc định, Zeek ghi tất cả thông tin này vào các tệp nhật ký được phân tách bằng tab hoặc JSON có cấu trúc tốt phù hợp để xử lý hậu kỳ bằng phần mềm bên ngoài. Người dùng cũng có thể chọn để cơ sở dữ liệu bên ngoài hoặc các sản phẩm SIEM sử dụng, lưu trữ, xử lý và trình bày dữ liệu để truy vấn.

Ngoài các bản ghi, Zeek đi kèm với chức năng tích hợp cho một loạt các nhiệm vụ phân tích và phát hiện, bao gồm trích xuất các tệp từ các phiên HTTP, phát hiện phần mềm độc hại bằng cách giao tiếp với các cơ quan đăng ký bên ngoài, báo cáo các phiên bản phần mềm dễ bị tấn công trên mạng, xác định các trang web phổ biến ứng dụng, phát hiện hành vi cưỡng bức SSH, xác thực chuỗi chứng chỉ SSL và hơn thế nữa.

Ngoài việc vận chuyển chức năng mạnh mẽ “out of box”, Zeek là một nền tảng hoàn toàn có thể tùy chỉnh và có thể mở rộng để phân tích lưu lượng truy cập. Zeek cung cấp cho người dùng một ngôn ngữ kịch bản hoàn chỉnh, dành riêng cho miền để thể hiện các tác vụ phân tích tùy ý. Hãy nghĩ về ZeekScripts như một “Python dành riêng cho miền” (hoặc Perl): giống như Python, hệ thống đi kèm với một tập hợp lớn các chức năng được tạo sẵn (“thư viện tiêu chuẩn”), nhưng người dùng cũng có thể sử dụng Zeek theo những cách mới lạ bằng cách viết mã tùy chỉnh. Thật vậy, tất cả các phân tích mặc định của Zeek, bao gồm ghi nhật ký, đều được thực hiện thông qua các tập lệnh; không có phân tích cụ thể nào được mã hóa cứng thành cốt lõi của hệ thống.

Zeek chạy trên phần cứng thiết bị và do đó cung cấp một giải pháp thay thế chi phí thấp cho các giải pháp độc quyền đắt tiền. Theo nhiều cách, Zeek vượt quá khả

năng của các công cụ giám sát mạng khác, những công cụ này thường bị giới hạn trong một nhóm nhỏ các nhiệm vụ phân tích được mã hóa cứng. Zeek không phải là một hệ thống phát hiện xâm nhập dựa trên chữ ký cổ điển (IDS); trong khi nó cũng hỗ trợ chức năng tiêu chuẩn như vậy, ZeekScripts tạo điều kiện cho nhiều phương pháp tiếp cận rất khác nhau để tìm ra hoạt động độc hại. Chúng bao gồm phát hiện lạm dụng ngữ nghĩa, phát hiện bất thường và phân tích hành vi.

Nhiều trang web triển khai Zeek để bảo vệ cơ sở hạ tầng của họ, bao gồm nhiều trường đại học, phòng nghiên cứu, trung tâm siêu máy tính, cộng đồng khoa học mở, các tập đoàn lớn và cơ quan chính phủ. Zeek đặc biệt nhắm mục tiêu theo dõi mạng tốc độ cao, khối lượng lớn và ngày càng nhiều trang web đang sử dụng hệ thống để giám sát mạng 10GE của họ, với một số đã chuyển sang liên kết 100GE

Zeek đáp ứng các cài đặt hiệu suất cao bằng cách hỗ trợ cân bằng tải có thể mở rộng. Các trang web lớn thường chạy "Zeek Cluster" trong đó bộ cân bằng tải front end tốc độ cao phân phối lưu lượng trên một số lượng PC end thích hợp, tất cả đều chạy các phiên bản Zeek chuyên dụng trên các lát lưu lượng riêng lẻ của chúng. Hệ thống quản lý trung tâm điều phối quá trình, đồng bộ hóa trạng thái trên các đầu sau và cung cấp cho người vận hành giao diện quản lý trung tâm để cấu hình và truy cập vào nhật ký tổng hợp. Khung quản lý tích hợp của Zeek, ZeekControl, hỗ trợ các thiết lập cụm như vậy.

Các tính năng cụm của Zeek hỗ trợ thiết lập đơn hệ thống và đa hệ thống. Đó là một phần lợi thế về khả năng mở rộng của Zeek. Ví dụ: quản trị viên có thể mở rộng quy mô Zeek trong một hệ thống càng lâu càng tốt và sau đó thêm các hệ thống khác một cách minh bạch khi cần thiết.

Tóm lại, Zeek được tối ưu hóa để diễn giải lưu lượng mạng và tạo nhật ký dựa trên lưu lượng đó. Nó không được tối ưu hóa để đối sánh byte và người dùng tìm kiếm các phương pháp phát hiện chữ ký sẽ được phục vụ tốt hơn bằng cách thử các hệ thống phát hiện xâm nhập như Suricata. Zeek cũng không phải là một công cụ phân tích giao thức theo nghĩa của Wireshark, tìm cách mô tả mọi yếu tố của lưu lượng mạng ở cấp khung hoặc một hệ thống lưu trữ lưu lượng ở dạng bắt gói (PCAP). Thay vào đó, Zeek nằm ở "phương tiện hài lòng" thể hiện nhật ký mạng nhỏ gọn nhưng có độ trung thực cao, giúp hiểu rõ hơn về lưu lượng mạng và việc sử dụng.

Zeek cung cấp nhiều lợi thế cho các nhóm bảo mật và mạng muốn hiểu rõ hơn về cách cơ sở hạ tầng của họ đang được sử dụng.

Các nhóm bảo mật thường phụ thuộc vào bốn loại nguồn dữ liệu khi cố gắng phát hiện và phản ứng với hoạt động đáng ngờ và độc hại. Chúng bao gồm các nguồn của bên thứ ba như cơ quan thực thi pháp luật, đồng nghiệp và các tổ chức tình báo về mối đe dọa thương mại hoặc phi lợi nhuận; dữ liệu mạng; cơ sở hạ tầng và dữ liệu ứng dụng, bao gồm nhật ký từ môi trường đám mây; và dữ liệu điểm cuối . Zeek chủ yếu là một nền tảng để thu thập và phân tích dạng dữ liệu thứ hai - dữ liệu mạng. Tuy nhiên, cả bốn đều là những yếu tố quan trọng trong chương trình của bất kỳ đội bảo mật nào.

Khi xem xét dữ liệu thu được từ mạng, có bốn loại dữ liệu có sẵn cho các nhà phân tích. Theo định nghĩa của mô hình giám sát an ninh mạng, bốn loại dữ liệu này là nội dung đầy đủ , dữ liệu giao dịch , nội dung trích xuất và dữ liệu cảnh báo . Sử dụng các kiểu dữ liệu này, người ta có thể ghi lại lưu lượng truy cập, tóm tắt lưu lượng truy cập, trích xuất lưu lượng truy cập (hoặc có lẽ chính xác hơn, trích xuất nội dung dưới dạng tệp) và đánh giá lưu lượng truy cập, tương ứng.

Việc thu thập và phân tích bốn loại dữ liệu giám sát an ninh mạng là rất quan trọng. Câu hỏi trở thành một trong những xác định cách tốt nhất để thực hiện mục tiêu này. Rất may, Zeek với tư cách là một nền tảng NSM cho phép thu thập ít nhất hai và theo một cách nào đó là ba, trong số các dạng dữ liệu này, cụ thể là dữ liệu giao dịch, nội dung được trích xuất và dữ liệu cảnh báo.

Zeek được biết đến nhiều nhất với dữ liệu giao dịch của nó. Theo mặc định, khi được chạy và được yêu cầu xem giao diện mạng, Zeek sẽ tạo ra một tập hợp các nhật ký giao dịch nhỏ gọn, có độ trung thực cao, được chú thích phong phú. Các nhật ký này mô tả các giao thức và hoạt động được nhìn thấy trên dây, theo cách không phán xét, trung lập về chính sách. Tài liệu này sẽ dành một lượng thời gian đáng kể để mô tả các tệp nhật ký Zeek phổ biến nhất để người đọc cảm thấy thoải mái với định dạng và học cách áp dụng chúng vào môi trường của họ.

Zeek cũng có thể dễ dàng xử lý các tệp từ lưu lượng mạng, nhờ vào khả năng trích xuất tệp của nó. Sau đó, các nhà phân tích có thể gửi các tệp đó đến hộp cát thực thi hoặc các công cụ kiểm tra tệp khác để điều tra bổ sung. Zeek có một số khả năng thực hiện phát hiện xâm nhập tập trung vào byte cổ điển, nhưng công việc đó phù hợp nhất với các gói như công cụ nguồn mở Snort hoặc Suricata. Tuy nhiên, Zeek có các khả năng khác có khả năng đưa ra các phán đoán dưới dạng cảnh báo, thông qua cơ chế thông báo của nó.

Zeek không được tối ưu hóa để ghi lưu lượng vào đĩa theo tinh thần thu thập dữ liệu nội dung đầy đủ và tác vụ đó được xử lý tốt nhất bởi phần mềm được viết để đáp ứng yêu cầu đó.

Ngoài các dạng dữ liệu mạng mà Zeek có thể thu thập và tạo ra một cách tự nhiên, Zeek có những lợi thế xuất hiện trong Zeek là gì? tiết diện. Chúng bao gồm chức năng tích hợp của nó cho một loạt các nhiệm vụ phân tích và phát hiện cũng như trạng thái của nó là một nền tảng có thể tùy chỉnh và mở rộng hoàn toàn để phân tích lưu lượng truy cập. Zeek cũng hấp dẫn vì khả năng chạy trên phần cứng hàng hóa, cho phép người dùng thuộc mọi đối tượng ít nhất có thể dùng thử Zeek với chi phí thấp.

https://docs.zeek.org/en/master/index.html

II. Zeek Scripts

1. Tập lệnh

Zeek bao gồm một ZeekScripts hướng sự kiện cung cấp phương tiện chính cho một tổ chức để mở rộng và tùy chỉnh chức năng của Zeek. Trên thực tế, hầu như tất cả đầu ra do Zeek tạo ra đều được tạo bởi Zeek scripts. Gần như dễ dàng hơn khi coi Zeek là một thực thể xử lý các kết nối hậu trường và tạo ra các sự kiện trong khi ZeekScripts là phương tiện mà thông qua đó chúng ta bình thường có thể giao tiếp. Các tập lệnh Zeek thông báo một cách hiệu quả cho Zeek rằng nếu có một sự kiện thuộc loại mà chúng tôi xác định, sau đó cho chúng tôi biết thông tin về kết nối để chúng tôi có thể thực hiện một số chức năng trên đó.

Thông thường, dễ hiểu nhất về ngôn ngữ ZeekScripts bằng cách xem một tập lệnh hoàn chỉnh và chia nhỏ thành các thành phần có thể nhận dạng được. Trong ví dụ này, chúng ta sẽ xem xét cách Zeek kiểm tra hàm băm SHA1 của các tệp khác nhau được trích xuất từ lưu lượng mạng dựa trên **Team Cymru Malware hash registry**. Team Cymru cũng điền vào bản ghi TXT của các phản hồi DNS của họ với cả dấu thời gian "được nhìn thấy lần đầu" và "tỷ lệ phát hiện" bằng số. Khía cạnh quan trọng cần hiểu là Zeek đã tạo hàm băm cho các tệp thông qua Files , framework, nhưng đó là policy/frameworks/files/detect-MHR.zeek chịu trách nhiệm tạo bản tra cứu DNS thích hợp, phân tích cú pháp phản hồi và tạo thông báo thích hợp

```
@load base/frameworks/files
@load base/frameworks/notice
@load frameworks/files/hash-all-files

module TeamCymruMalwareHashRegistry;

export {
    redef enum Notice::Type += {
        ## The hash value of a file transferred over HTTP matched in the
        ## malware hash registry.
        Match
    };

    ## File types to attempt matching against the Malware Hash Registry.
    option match_file_types = /application\/x-dosexec/ |
                              /application\/vnd.ms-cab-compressed/ |
                              /application\/pdf/ |
                              /application\/x-shockwave-flash/ |
                              /application\/x-java-applet/ |
                              /application\/jar/ |
                              /video\/mp4/;

    ## The Match notice has a sub message with a URL where you can get more
    ## information about the file. The %s will be replaced with the SHA-1
    ## hash of the file.
    option match_sub_url = "https://www.virustotal.com/en/search/?query=%s";

    ## The malware hash registry runs each malware sample through several
    ## A/V engines.  Team Cymru returns a percentage to indicate how
    ## many A/V engines flagged the sample as malicious. This threshold
    ## allows you to require a minimum detection rate.
    option notice_threshold = 10;
}

function do_mhr_lookup(hash: string, fi: Notice::FileInfo)
    {
    local hash_domain = fmt("%s.malware.hash.cymru.com", hash);

    when ( local MHR_result = lookup_hostname_txt(hash_domain) )
        {
        # Data is returned as "<dateFirstDetected> <detectionRate>"
        local MHR_answer = split_string1(MHR_result, / /);

        if ( |MHR_answer| == 2 )
            {
            local mhr_detect_rate = to_count(MHR_answer[1]);

            if ( mhr_detect_rate >= notice_threshold )
                {
                local mhr_first_detected = double_to_time(to_double(MHR_answer[0]));
                local readable_first_detected = strftime("%Y-%m-%d %H:%M:%S", mhr_first_detected);
                local message = fmt("Malware Hash Registry Detection rate: %d%%  Last seen: %s", mhr_det
                local virustotal_url = fmt(match_sub_url, hash);
                # We don't have the full fa_file record here in order to
                # avoid the "when" statement cloning it (expensive!).
                local n: Notice::Info = Notice::Info($note=Match, $msg=message, $sub=virustotal_url);
                Notice::populate_file_info2(fi, n);
                NOTICE(n);
                }
            }
        }
    }

event file_hash(f: fa_file, kind: string, hash: string)
    {
    if ( kind == "sha1" && f?$info && f$info?$mime_type &&
         match_file_types in f$info$mime_type )
        do_mhr_lookup(hash, Notice::create_file_info(f));
    }
```

Về mặt trực quan, có ba phần riêng biệt của kịch bản. Đầu tiên, có một mức cơ sở không có thụt lề trong đó các thư viện được bao gồm trong tập lệnh thông

qua @load và một không gian tên được xác định với module. Tiếp theo là phần được định dạng và thụt lề giải thích các biến tùy chỉnh đang được cung cấp (export) như một phần của không gian tên của tập lệnh. Cuối cùng, có một phần được định dạng và thụt lề thứ hai mô tả các hướng dẫn cần thực hiện cho một sự kiện cụ thể ( ). Đừng nản lòng nếu bạn không hiểu từng phần của kịch bản; chúng tôi sẽ trình bày những điều cơ bản về script và nhiều hơn thế nữa trong các phần sau.event file_hash.

```
@load base/frameworks/files
@load base/frameworks/notice
@load frameworks/files/hash-all-files
```

Phần đầu tiên của script bao gồm các lệnh @load xử lý __load__.zeekscript trong các thư mục tương ứng đang được tải. Các @load chỉ thị thường được coi là thực hành tốt hoặc thậm chí chỉ là cách cư xử tốt khi viết các tập lệnh Zeek để đảm bảo chúng có thể được sử dụng riêng. Mặc dù không chắc trong quá trình triển khai sản xuất đầy đủ của Zeek, các tài nguyên bổ sung này sẽ không được tải, nhưng không phải là một thói quen xấu nếu bạn cố gắng tham gia khi bạn có nhiều kinh nghiệm hơn với Zeek scripting. Nếu bạn mới bắt đầu, mức độ chi tiết này có thể không hoàn toàn cần thiết. Các @load chỉ thị đảm bảo khung Tệp, khung Thông báo và tập lệnh băm tất cả các tệp đã được tải bởi Zeek.

```
export {
    redef enum Notice::Type += {
        ## The hash value of a file transferred over HTTP matched in the
        ## malware hash registry.
        Match
    };

    ## File types to attempt matching against the Malware Hash Registry.
    option match_file_types = /application\/x-dosexec/ |
                              /application\/vnd.ms-cab-compressed/ |
                              /application\/pdf/ |
                              /application\/x-shockwave-flash/ |
                              /application\/x-java-applet/ |
                              /application\/jar/ |
                              /video\/mp4/;

    ## The Match notice has a sub message with a URL where you can get more
    ## information about the file. The %s will be replaced with the SHA-1
    ## hash of the file.
    option match_sub_url = "https://www.virustotal.com/en/search/?query=%s";

    ## The malware hash registry runs each malware sample through several
    ## A/V engines.  Team Cymru returns a percentage to indicate how
    ## many A/V engines flagged the sample as malicious. This threshold
    ## allows you to require a minimum detection rate.
    option notice_threshold = 10;
}
```

Phần **export** xác định lại một hằng số mô tả loại thông báo mà chúng tôi sẽ tạo với khung Thông báo. Zeek cho phép các hằng số có thể xác định lại, thoạt đầu, có vẻ hơi phản trực quan. Chúng ta sẽ tìm hiểu sâu hơn về các hằng số trong chương sau, hiện tại, hãy nghĩ về chúng như những biến chỉ có thể được thay đổi trước khi Zeek bắt đầu chạy. Bằng cách mở rộng **Notice::Type** như được hiển thị, điều này cho phép hàm **NOTICE** tạo ra các thông báo với trường $note được chỉ định là TeamCymruMalwareHashRegistry::Match. Notices cho phép Zeek tạo một số loại thông báo bổ sung ngoài các loại nhật ký mặc định của nó. Thông thường, thông báo bổ sung này ở dạng email được tạo và gửi đến một địa chỉ được định cấu hình trước, nhưng có thể được thay đổi tùy thuộc vào nhu cầu của việc triển khai. Phần export kết thúc bằng một vài định nghĩa về constant liệt kê loại tệp chúng tôi muốn đối sánh và tỷ lệ phần trăm ngưỡng phát hiện tối thiểu mà chúng tôi quan tâm.

Cho đến thời điểm này, tập lệnh chỉ thực hiện một số thiết lập cơ bản. Với phần tiếp theo, tập lệnh bắt đầu xác định các hướng dẫn để thực hiện trong một sự kiện nhất định.

```
function do_mhr_lookup(hash: string, fi: Notice::FileInfo)
    {
    local hash_domain = fmt("%s.malware.hash.cymru.com", hash);

    when ( local MHR_result = lookup_hostname_txt(hash_domain) )
        {
        # Data is returned as "<dateFirstDetected> <detectionRate>"
        local MHR_answer = split_string1(MHR_result, / /);

        if ( |MHR_answer| == 2 )
            {
            local mhr_detect_rate = to_count(MHR_answer[1]);

            if ( mhr_detect_rate >= notice_threshold )
                {
                local mhr_first_detected = double_to_time(to_double(MHR_answer[0]));
                local readable_first_detected = strftime("%Y-%m-%d %H:%M:%S", mhr_first_detected);
                local message = fmt("Malware Hash Registry Detection rate: %d%%  Last seen: %s", mhr_det
                local virustotal_url = fmt(match_sub_url, hash);
                # We don't have the full fa_file record here in order to
                # avoid the "when" statement cloning it (expensive!).
                local n: Notice::Info = Notice::Info($note=Match, $msg=message, $sub=virustotal_url);
                Notice::populate_file_info2(fi, n);
                NOTICE(n);
                }
            }
        }
    }

event file_hash(f: fa_file, kind: string, hash: string)
    {
    if ( kind == "sha1" && f?$info && f$info?$mime_type &&
         match_file_types in f$info$mime_type )
        do_mhr_lookup(hash, Notice::create_file_info(f));
```

Workhorse của script được chứa trong trình xử lý của sự kiện cho file_hash. Sự file_hash này cho phép các tập lệnh truy cập thông tin được liên kết với một tệp mà khung phân tích tệp của Zeek đã tạo một hàm băm. Trình xử lý sự kiện được chuyển chính tệp dưới dạng f, loại thuật toán thông báo được sử dụng dưới dạng kind và hàm băm được tạo dưới dạng hash.

Trong trình xử lý sự kiện file_hash, có một câu lệnh if được sử dụng để kiểm tra loại băm chính xác, trong trường hợp này là băm SHA1.Nó cũng kiểm tra kiểu mime type mà đã được xác định là được chú ý như được định nghĩa trong constant match_file_types . So sánh được thực hiện dựa trên biểu thức f$info$mime_type, sử dụng toán tử $ tham chiếu để kiểm tra giá trị mime_type bên trong biến f$info. Nếu toàn bộ biểu thức đánh giá là true, thì một hàm phụ được gọi để thực hiện phần còn lại của công việc. Trong hàm đó, một biến cục bộ được định nghĩa để chứa một chuỗi bao gồm hàm băm SHA1 được nối với .malware.hash.cymru.com; giá trị này sẽ là miền được truy vấn trong **the malware hash registry** (sổ đăng ký băm phần mềm độc hại)

Phần còn lại của tập lệnh được chứa trong một khối when. Nói tóm lại, một khối when được sử dụng khi Zeek cần thực hiện các hành động không đồng bộ, chẳng hạn như tra cứu DNS, để đảm bảo rằng hiệu suất không bị ảnh hưởng. Khối when thực hiện tra cứu TXT DNS và lưu trữ kết quả trong biến cục bộ MHR_result. Về mặt hiệu quả, quá trình xử lý sự kiện này vẫn tiếp tục và khi nhận được các giá trị được trả về **lookup_hostname_txt**, khối when sẽ được thực thi. Khối when chia chuỗi được trả về thành một phần cho vào ngày mà phần mềm độc hại được phát hiện lần đầu tiên và tỷ lệ phát hiện, bằng cách tách văn bản trên không gian và lưu trữ các giá trị được trả về trong một biến bảng cục bộ. Trong hàm do_mhr_lookup, nếu bảng trả về split1 có hai mục nhập, cho biết sự phân tách thành công, lưu trữ ngày phát hiện trong mhr_first_detected và tỷ lệ mhr_detect_rate sử dụng các chức năng chuyển đổi thích hợp. Kể từ thời điểm này, Zeek biết rằng đã thấy một tệp được truyền có chứa hàm băm đã được tìm thấy bởi Team Cymru Malware Hash Registry , phần còn lại của tập lệnh được dành để tạo thông báo.

Thời gian phát hiện được xử lý thành một chuỗi biểu diễn và được lưu trữ trong readable_first_detected. Sau đó, tập lệnh so sánh tỷ lệ phát hiện với tỷ lệ notice_threshold đã được xác định trước đó. Nếu tỷ lệ phát hiện đủ cao, tập lệnh sẽ tạo mô tả ngắn gọn về thông báo và lưu trữ trong biến message. Nó cũng tạo ra

một URL khả thi để kiểm tra mẫu dựa virustotal.com trên cơ sở dữ liệu của và thực hiện lệnh gọi để **NOTICE** chuyển thông tin liên quan vào khung Thông báo.

Trong khoảng vài chục dòng mã, Zeek cung cấp một tiện ích tuyệt vời mà khó có thể thực hiện và triển khai với các sản phẩm khác. Trên thực tế, tuyên bố rằng Zeek làm điều này trong một số ít dòng như vậy là một hướng đi sai lầm; có rất nhiều thứ thực sự đang diễn ra ở hậu trường trong Zeek, nhưng chính việc bao gồm ngôn ngữ kịch bản sẽ cho phép các nhà phân tích truy cập vào các lớp bên dưới đó một cách ngắn gọn và được xác định rõ ràng.

2. Trình xử lý sự kiện

ZeekScripts được điều khiển bởi sự kiện, đây là một sự thay đổi cần thiết so với phần lớn các ngôn ngữ lập trình kịch bản mà hầu hết người dùng sẽ có kinh nghiệm trước đó. Việc viết kịch bản trong Zeek phụ thuộc vào việc xử lý các sự kiện do Zeek tạo ra khi nó xử lý lưu lượng mạng, thay đổi trạng thái của cấu trúc dữ liệu thông qua các sự kiện đó và đưa ra quyết định về thông tin được cung cấp. Cách tiếp cận tập lệnh này thường có thể gây nhầm lẫn cho người dùng đến với Zeek từ ngôn ngữ thủ tục hoặc chức năng, nhưng một khi cú sốc ban đầu hết, nó sẽ trở nên rõ ràng hơn với mỗi lần tiếp xúc.

Cốt lõi của Zeek hoạt động để đặt các sự kiện vào một “event queue”(hàng đợi sự kiện” có thứ tự, cho phép người xử lý sự kiện xử lý chúng trên cơ sở ai đến trước được phục vụ trước. Trên thực tế, đây là chức năng cốt lõi của Zeek vì nếu không có các tập lệnh được viết để thực hiện các hành động rời rạc trên các sự kiện, sẽ có rất ít hoặc không có đầu ra có thể sử dụng được. Như vậy, hiểu biết cơ bản về hàng đợi sự kiện, các sự kiện đang được tạo và cách mà trình xử lý sự kiện xử lý các sự kiện đó là cơ sở để không chỉ học viết script cho Zeek mà còn để hiểu về chính Zeek.

Làm quen với các sự kiện cụ thể do Zeek tạo ra là một bước tiến lớn để xây dựng tư duy làm việc với Zeek script. Phần lớn các sự kiện do Zeek tạo ra được xác định trong hàm tích hợp (`*.bif`) các tệp cũng đóng vai trò là cơ sở cho tài liệu sự kiện trực tuyến. Những nhận xét nội dòng này được tổng hợp thành một hệ thống tài liệu trực tuyến sử dụng Zeekygen. Cho dù bắt đầu một tập lệnh từ đầu hay đọc và duy trì tập lệnh của người khác, việc có sẵn các định nghĩa sự kiện tích hợp sẵn là một nguồn tài nguyên tuyệt vời cần có trong tay. Đối với bản phát hành 2.0, các nhà phát triển Zeek đã nỗ lực đáng kể trong việc tổ chức và lập hồ sơ cho mọi sự kiện. Nỗ lực này dẫn đến các tệp tích hợp trong hàm được tổ chức sao cho mỗi mục nhập chứa

tên sự kiện mô tả, các đối số được truyền cho sự kiện và giải thích ngắn gọn về các hàm được sử dụng.

```
## Generated for DNS requests. For requests with multiple queries, this event
## is raised once for each.
##
## See `Wikipedia <http://en.wikipedia.org/wiki/Domain_Name_System>`__ for more
## information about the DNS protocol. Zeek analyzes both UDP and TCP DNS
## sessions.
##
## c: The connection, which may be UDP or TCP depending on the type of the
##    transport-layer session being analyzed.
##
## msg: The parsed DNS message header.
##
## query: The queried name.
##
## qtype: The queried resource record type.
##
## qclass: The queried resource record class.
##
## .. zeek:see:: dns_AAAA_reply dns_A_reply dns_CNAME_reply dns_EDNS_addl
##    dns_HINFO_reply dns_MX_reply dns_NS_reply dns_PTR_reply dns_SOA_reply
##    dns_SRV_reply dns_TSIG_addl dns_TXT_reply dns_WKS_reply dns_end
##    dns_full_request dns_mapping_altered dns_mapping_lost_name dns_mapping_new_name
##    dns_mapping_unverified dns_mapping_valid dns_message dns_query_reply
##    dns_rejected non_dns_request dns_max_queries dns_session_timeout dns_skip_addl
##    dns_skip_all_addl dns_skip_all_auth dns_skip_auth
event dns_request%(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count%);
```

Trên đây là một đoạn của tài liệu cho sự kiện **dns_request** .Nó được tổ chức sao cho documentation, commentary và list các đối số đứng trước định nghĩa sự kiện thực tế được Zeek sử dụng. Khi Zeek phát hiện các yêu cầu DNS được cấp bởi một người khởi tạo, nó sẽ đưa ra sự kiện này và bất kỳ số lượng tập lệnh nào sau đó có quyền truy cập vào dữ liệu mà Zeek chuyển cùng với sự kiện. Trong ví dụ này, Zeek không chỉ chuyển thông báo, truy vấn, loại truy vấn và lớp truy vấn cho yêu cầu DNS mà còn chuyển một bản ghi được sử dụng cho chính kết nối.

3. Các kiểu dữ liệu và Cấu trúc dữ liệu

- **Scope**

Khai báo các biến trong Zeek có hai dạng. Các biến có thể được khai báo có hoặc không có định nghĩa ở dạng hoặc tương ứng; mỗi trong số đó tạo ra cùng một kết quả nếu được đánh giá cùng loại với . Quyết định sử dụng loại nào có thể được

quyết định bởi sở thích cá nhân và khả năng dễ đọc
SCOPE name: TYPESCOPE name = EXPRESSIONEXPRESSIONTYPE

```
event zeek_init()
    {
    local a: int;
    a = 10;
    local b = 10;

    if ( a == b )
        print fmt("A: %d, B: %d", a, b);
    }
```

- **Global Variables**

Biến toàn cục được sử dụng khi trạng thái của biến cần được theo dõi, không có gì đáng ngạc nhiên, trên toàn cầu. Mặc dù có một số lưu ý, khi một tập lệnh khai báo một biến bằng cách sử dụng phạm vi toàn cục, thì tập lệnh đó sẽ cấp quyền truy cập vào biến đó từ các tập lệnh khác. Tuy nhiên, khi một tập lệnh sử dụngmoduletừ khóa để cung cấp cho tập lệnh một không gian tên, cần phải chú ý hơn đến việc khai báo các khối cầu để đảm bảo kết quả dự kiến. Khi một toàn cục được khai báo trong một tập lệnh với một không gian tên, có hai kết quả có thể xảy ra. Đầu tiên, biến chỉ có sẵn trong ngữ cảnh của không gian tên. Trong trường hợp này, các tập lệnh khác trong cùng một không gian tên sẽ có quyền truy cập vào biến được khai báo trong khi các tập lệnh sử dụng một không gian tên khác hoặc không có không gian tên hoàn toàn sẽ không có quyền truy cập vào biến. Ngoài ra, nếu một biến toàn cục được khai báo trong một khối thì biến đó có sẵn cho bất kỳ tập lệnh nào khác thông qua quy ước đặt tên , tức là biến đó cần được "xác định phạm vi" theo tên của mô-đun mà nó được khai báo.export { ... }<module name>::<variable name>

Khi từ khóa module được sử dụng trong một tập lệnh, các biến được khai báo được cho là nằm trong "không gian tên" của mô-đun đó. Trong trường hợp một biến toàn cục có thể được truy cập chỉ bằng tên của nó khi nó không được khai báo trong một mô-đun, thì một biến toàn cục được khai báo trong một mô-đun phải được xuất và sau đó được truy cập qua .<module name>::<variable name>

- **Constants**

Zeek cũng sử dụng các hằng số, được biểu thị bằng const từ khóa. Không giống như global values, constants chỉ có thể được đặt hoặc thay đổi tại thời điểm phân

tích cú pháp nếu &redef thuộc tính đã được sử dụng. Sau đó (trong thời gian chạy), các constant không thể thay đổi được. Trong hầu hết các trường hợp,các constant có thể xác định lại được sử dụng trong các tập lệnh Zeek làm vùng chứa cho các tùy chọn cấu hình. Ví dụ: tùy chọn cấu hình để ghi mật khẩu được giải mã từ các luồng HTTP được lưu trữ **HTTP::default_capture_password** như được hiển thị trong đoạn trích rút gọn từ base / protocols / http / main.zeek bên dưới.

```
module HTTP;

export {
    ## This setting changes if passwords used in Basic-Auth are captured or
    ## not.
    const default_capture_password = F &redef;
}
```

Vì const đã được khai báo cùng với &redef thuộc tính, nên nếu chúng ta cần bật tùy chọn này trên toàn cục,có thể làm như vậy bằng cách thêm dòng sau vào file site/local.zeek của trước khi kích hoạt Zeek.

```
@load base/protocols/http

redef HTTP::default_capture_password = T;
```

Mặc dù ý tưởng về const có thể xác định lại có thể là khó khăn, nhưng ràng buộc rằng hằng số chỉ có thể được thay đổi tại thời điểm phân tích cú pháp ngay cả với thuộc tính &redef. Trong đoạn mã dưới đây, một bảng các chuỗi được lập chỉ mục bởi các cổng được khai báo là một hằng số trước khi hai giá trị được thêm vào bảng thông qua các câu lệnh redef. Bảng sau đó được in trong một `zeek_init` sự kiện. Nếu chúng tôi cố gắng thay đổi bảng trong trình xử lý sự kiện, Zeek sẽ thông báo cho người dùng về lỗi và tập lệnh sẽ không thành công.

data_type_const.zeek

```
const port_list: table[port] of string &redef;

redef port_list += { [6666/tcp] = "IRC"};
redef port_list += { [80/tcp] = "WWW" };

event zeek_init()
    {
    print port_list;
    }
```

- **Local Variables**

Trong khi global value và constant value có sẵn rộng rãi trong scriptland thông qua nhiều phương tiện khác nhau, khi một biến được xác định với phạm vi cục bộ, tính khả dụng của nó bị hạn chế đối với phần thân của sự kiện hoặc hàm mà nó được khai báo. Các biến cục bộ có xu hướng được sử dụng cho các giá trị chỉ cần thiết trong một phạm vi cụ thể và một khi quá trình xử lý tập lệnh vượt ra ngoài phạm vi đó và không còn được sử dụng nữa, biến đó sẽ bị xóa. Zeek duy trì tên của người dân địa phương tách biệt với những người có thể nhìn thấy trên toàn cầu, một ví dụ về số đó được minh họa bên dưới.

```
function add_two(i: count): count
    {
    local added_two = i+2;
    print fmt("i + 2 = %d", added_two);
    return added_two;
    }

event zeek_init()
    {
    local test = add_two(10);
    }
```

Tập lệnh thực thi trình xử lý sự kiện **zeek_init** mà lần lượt gọi hàm add_two(i:count) với đối số là 10. Khi Zeek nhập hàm add_two, nó cung cấp một biến phạm vi cục bộ được gọi added_two để giữ giá trị i+2, trong trường hợp này 12 ,. Sau đó, hàm add_two in ra giá trị của biến và trả về giá trị của nó cho trình xử lý sự kiện zeek_init. Tại thời điểm này, biến đã hết phạm vi added_two và không còn tồn tại trong khi giá trị 12 vẫn được sử dụng và được lưu trữ trong biến phạm vi

cục bộ test . Khi Zeek kết thúc quá trình xử lý hàm zeek_init, biến được gọi không còn trong phạm vi và vì không tồn tại các tham chiếu khác đến giá trị 12, nên giá trị cũng bị xóa.

- **Data Structures**

Khá khó để nói về các kiểu dữ liệu của Zeek một cách thực tế mà không đề cập đến các cấu trúc dữ liệu có sẵn trong Zeek. Một số đặc điểm thú vị hơn của kiểu dữ liệu được tiết lộ khi được sử dụng bên trong cấu trúc dữ liệu, nhưng do cấu trúc dữ liệu được tạo thành từ các kiểu dữ liệu, nên nó chuyển thành bài toán "gà và trứng" khá nhanh. Do đó, chúng tôi sẽ giới thiệu các kiểu dữ liệu từ chế độ xem chim trước khi đi sâu vào cấu trúc dữ liệu và từ đó khám phá đầy đủ hơn về các kiểu dữ liệu.

Bảng dưới đây cho thấy các loại nguyên tử được sử dụng trong Zeek, trong đó bốn loại đầu tiên sẽ có vẻ quen thuộc nếu bạn có một số kinh nghiệm viết kịch bản, trong khi sáu loại còn lại ít phổ biến hơn ở các ngôn ngữ khác. Sẽ không có gì ngạc nhiên khi ngôn ngữ kịch bản cho nền tảng Giám sát an ninh mạng có một tập hợp các kiểu dữ liệu tập trung vào mạng khá mạnh mẽ.

Loại dữ liệu	Sự mô tả
`int`	Số nguyên có dấu 64 bit
`count`	Số nguyên không dấu 64 bit
`double`	độ chính xác nổi kép chính xác
`bool`	boolean (T / F)
`addr`	Địa chỉ IP, IPv4 và IPv6
`port`	cổng lớp vận chuyển
`subnet`	Mặt nạ mạng con CIDR
`time`	thời gian kỷ nguyên tuyệt đối
`interval`	một khoảng thời gian
`pattern`	biểu hiện thông thường

- **Sets**

Các tập hợp trong Zeek được sử dụng để lưu trữ các phần tử duy nhất của cùng một kiểu dữ liệu. Về bản chất, bạn có thể coi chúng là "một tập hợp số nguyên duy nhất" hoặc "một tập hợp địa chỉ IP duy nhất". Mặc dù việc khai báo một tập hợp có thể khác nhau dựa trên kiểu dữ liệu được thu thập, tập hợp sẽ luôn chứa các phần tử duy nhất và các phần tử trong tập hợp sẽ luôn có cùng một kiểu dữ liệu. Các yêu cầu như vậy làm cho kiểu dữ liệu tập hợp trở nên hoàn hảo cho thông tin vốn dĩ là duy nhất, chẳng hạn như cổng hoặc địa chỉ IP. Đoạn mã bên dưới hiển thị cả khai báo rõ ràng và ẩn ý về một tập hợp có phạm vi cục bộ.

```
event zeek_init()
    {
    local ssl_ports: set[port];
    local non_ssl_ports = set( 23/tcp, 80/tcp, 143/tcp, 25/tcp );
    }
```

Các tập hợp được khai báo bằng cách sử dụng định dạng . Việc thêm và loại bỏ các phần tử trong một tập hợp được thực hiện bằng cách sử dụng câu lệnh and . Khi bạn đã chèn các phần tử vào tập hợp, có khả năng là bạn sẽ cần phải lặp lại tập hợp đó hoặc kiểm tra tư cách thành viên trong tập hợp, cả hai đều được bao phủ bởi toán tử. Trong trường hợp lặp qua một tập hợp, việc kết hợp câu lệnh và toán tử sẽ cho phép bạn xử lý tuần tự từng phần tử của tập hợp như hình dưới đây.SCOPE var_name: set[TYPE]adddeleteinforin

```
    for ( i in ssl_ports )
        print fmt("SSL Port: %s", i);

    for ( i in non_ssl_ports )
        print fmt("Non-SSL Port: %s", i);
```

Ở đây, for câu lệnh lặp qua nội dung của tập hợp lưu trữ từng phần tử trong biến tạm thời i. Với mỗi lần lặp của for vòng lặp, phần tử tiếp theo được chọn. Vì các tập hợp không phải là một kiểu dữ liệu có thứ tự, bạn không thể đảm bảo thứ tự của các phần tử khi for xử lý vòng lặp.

Để kiểm tra tư cách thành viên trong một tập hợp, in câu lệnh có thể được kết hợp với một if câu lệnh để trả về giá trị true hoặc false. Nếu phần tử chính xác trong điều kiện đã có trong tập hợp, điều kiện trả về true và phần thân sẽ thực thi. Câu in lệnh cũng có thể bị phủ định bởi ! toán tử để tạo ra nghịch đảo của điều kiện. Mặc

dù chúng tôi có thể viết lại dòng tương ứng bên dưới để cố gắng tránh sử dụng cấu trúc này; thay vào đó, phủ định chính toán tử in. Mặc dù chức năng giống nhau, nhưng việc sử dụng hiệu quả hơn cũng như cấu trúc tự nhiên hơn sẽ hỗ trợ khả năng đọc tập lệnh của bạn. if (!(587/tcp in ssl_ports))!in

```
13      # Check for SMTPS
14      if ( 587/tcp !in ssl_ports )
15          add ssl_ports[587/tcp];
```

- **Tables**

Một bảng trong Zeek là một ánh xạ của một khóa đến một giá trị hoặc lợi nhuận. Mặc dù các giá trị không nhất thiết phải là duy nhất, nhưng mỗi khóa trong bảng phải là duy nhất để duy trì ánh xạ 1-1 của các khóa với các giá trị.

```
1   event zeek_init()
2       {
3       # Declaration of the table.
4       local ssl_services: table[string] of port;
5
6       # Initialize the table.
7       ssl_services = table(["SSH"] = 22/tcp, ["HTTPS"] = 443/tcp);
8
9       # Insert one key-yield pair into the table.
10      ssl_services["IMAPS"] = 993/tcp;
11
12      # Check if the key "SMTPS" is not in the table.
13      if ( "SMTPS" !in ssl_services )
14          ssl_services["SMTPS"] = 587/tcp;
15
16      # Iterate over each key in the table.
17      for ( k in ssl_services )
18          print fmt("Service Name:  %s - Common Port: %s", k, ssl_services[k]);
19      }
```

```
$ zeek data_struct_table_declaration.zeek
Service Name:  SSH - Common Port: 22/tcp
Service Name:  HTTPS - Common Port: 443/tcp
Service Name:  SMTPS - Common Port: 587/tcp
Service Name:  IMAPS - Common Port: 993/tcp
```

- **Vector**

Vectơ thực hiện nhiều chức năng giống như mảng kết hợp với các số nguyên không dấu làm chỉ số của chúng. Tuy nhiên, chúng hiệu quả hơn thế và chúng cho phép truy cập có thứ tự. Vì vậy, bất cứ lúc nào bạn cần lưu trữ tuần tự dữ liệu cùng

loại, trong Zeek, bạn nên tiếp cận với một vectơ. Vectơ là một tập hợp các đối tượng, tất cả đều có cùng kiểu dữ liệu, các phần tử có thể được thêm vào hoặc loại bỏ động. Vì Vectơ sử dụng bộ nhớ liền kề cho các phần tử của chúng, nên nội dung của vectơ có thể được truy cập thông qua phần bù số được lập chỉ mục bằng 0.

Định dạng khai báo Vectơ tuân theo mẫu của các khai báo khác, cụ thể là tên vectơ của bạn ở đâu và là kiểu dữ liệu của các thành viên của nó. Ví dụ: đoạn mã sau đây cho thấy một khai báo rõ ràng và ẩn ý về hai vectơ có phạm vi cục bộ. Tập lệnh điền vectơ đầu tiên bằng cách chèn các giá trị vào cuối; nó thực hiện điều đó bằng cách đặt tên vectơ giữa hai đường ống thẳng đứng để lấy độ dài hiện tại của vectơ trước khi in nội dung của cả Vectơ và độ dài hiện tại của chúng. `SCOPE v: vector of TvT`

```
event zeek_init()
    {
    local v1: vector of count;
    local v2 = vector(1, 2, 3, 4);

    v1 += 1;
    v1 += 2;
    v1 += 3;
    v1 += 4;

    print fmt("contents of v1: %s", v1);
    print fmt("length of v1: %d", |v1|);
    print fmt("contents of v2: %s", v2);
    print fmt("length of v2: %d", |v2|);
    }
```

```
$ zeek data_struct_vector_declaration.zeek
contents of v1: [1, 2, 3, 4]
length of v1: 4
contents of v2: [1, 2, 3, 4]
length of v2: 4
```

4. Hook

Hook là một hàm khác có chung đặc điểm của cả 1 **function** và 1 event . Chúng giống như các **event** trong đó nhiều thân trình xử lý có thể được xác định cho cùng một định danh hook nối và thứ tự thực thi có thể được thực thi với. Chúng giống các hàm hơn theo cách chúng được gọi / gọi, bởi vì, không giống như các **sự kiện**, việc thực thi của chúng diễn ra ngay lập tức và chúng không được lên lịch thông qua hàng

đợi **sự kiện .** Ngoài ra, một tính năng độc đáo của hook là một phần thân của trình xử lý hook nhất định có thể làm ngắn mạch việc thực thi các trình xử lý hook còn lại chỉ đơn giản bằng cách thoát khỏi phần thân do một câu lệnh (trái ngược với một hoặc chỉ đạt đến phần cuối của phần thân). **event&prioritybreakreturn**

Một kiểu hook được khai báo như sau:

```
hook( argument* )
```

Vị trí `argument*` là danh sách các đối số được phân tách bằng dấu phẩy (có thể trống). Ví dụ:

```
global myhook: hook(s: string, vs: vector of string);
```

Tại đây `myhook` là mã định danh kiểu hook và chưa có cơ quan xử lý hook nào được xác định cho nó. Để xác định một số phần thân của trình xử lý hook, cú pháp trông như sau:

```
hook myhook(s: string, vs: vector of string) &priority=10
    {
    print "priority 10 myhook handler", s, vs;
    s = "bye";
    vs += "modified";
    }

hook myhook(s: string, vs: vector of string)
    {
    print "break out of myhook handling", s, vs;
    break;
    }

hook myhook(s: string, vs: vector of string) &priority=-5
    {
    print "not going to happen", s, vs;
    }
```

Lưu ý rằng không bắt buộc phải khai báo đầu tiên (chuyển tiếp) `myhook` dưới dạng hook. Các kiểu đối số phải khớp với tất cả các trình xử lý hook và bất kỳ khai báo chuyển tiếp nào của một hook đã cho.

Để gọi thực thi ngay lập tức tất cả các phần thân của trình xử lý hook, chúng được gọi tương tự như một hàm, ngoại trừ đứng trước hook từ khóa:

```
hook myhook("hi", vector("foo"));
```

hoặc

```
priority 10 myhook handler, hi, [foo]
break out of myhook handling, hi, [foo, modified]
```

Lưu ý rằng cách gán lại một hook đối số (trong ví dụ) sẽ không hiển thị với các trình xử lý còn lại, nhưng vẫn có thể sửa đổi các giá trị của các loại tổng hợp / tổng hợp như , hoặc . s = "bye"hookvectorrecordsettable

Giá trị trả về của một cuộc gọi hook là một giá trị ngầm định bool với T nghĩa là tất cả các trình xử lý cho hook đã được thực thi và F có nghĩa là chỉ một số trình xử lý có thể đã được thực thi do một phần thân của trình xử lý thoát ra do một break câu lệnh.

Hook cũng được phép có nhiều khai báo nguyên mẫu / thay thế, giống như một . event

5. Frameworks

Zeek bao gồm một số khung phần mềm cung cấp chức năng thường được sử dụng cho lớp kịch bản. Ngoài những thứ khác, các khung công tác này nâng cao khả năng của Zeek trong việc nhập dữ liệu, cấu trúc và lọc đầu ra của nó, điều chỉnh cài đặt trong thời gian chạy và tương tác với các thành phần khác trong mạng của bạn. Hầu hết các khuôn khổ bao gồm chức năng được triển khai trong lõi của Zeek, với các cấu trúc dữ liệu và API tương ứng được hiển thị trên lớp tập lệnh.

Một số khung công tác nhắm mục tiêu các trường hợp sử dụng tương đối cụ thể, trong khi những khung công tác khác chạy trong hầu hết mọi cài đặt Zeek. Ví dụ, khung ghi nhật ký cung cấp máy hook đằng sau tất cả các nhật ký Zeek đã đề cập

trước đó. Các khung công tác cũng xây dựng dựa trên nhau, vì vậy rất đáng để biết khả năng của chúng. Một số Framework sau:

- Framework nhật ký
- Framework thông báo
- Framework đầu vào
- Framework chữ ký
- Framework cấu hình
- Framework giám sát

❖ Framework nhật ký

Zeek đi kèm với giao diện ghi nhật ký dựa trên khóa-giá trị linh hoạt cho phép kiểm soát chi tiết những gì được ghi lại và cách nó được ghi lại. Tài liệu này mô tả cách ghi nhật ký có thể được tùy chỉnh và mở rộng.

Thuật ngữ

Giao diện ghi nhật ký của Zeek được xây dựng dựa trên ba điểm trừu tượng chính:

Streams

Một dòng nhật ký tương ứng với một nhật ký duy nhất. Nó xác định tập hợp các trường mà nhật ký bao gồm với tên và kiểu của chúng. Ví dụ như luồng kết nối để ghi tóm tắt kết nối và luồng http để ghi lại hoạt động HTTP.

Filters

Mỗi luồng có một tập hợp các bộ lọc được đính kèm để xác định thông tin nào được ghi ra ngoài và cách thức. Theo mặc định, mỗi luồng có một bộ lọc mặc định chỉ ghi mọi thứ trực tiếp vào đĩa. Tuy nhiên, các bộ lọc bổ sung có thể được thêm vào để chỉ ghi lại một tập hợp con của các bản ghi nhật ký, ghi vào các đầu ra khác nhau hoặc đặt khoảng thời gian xoay vòng tùy chỉnh. Nếu tất cả các bộ lọc bị xóa khỏi một luồng, thì đầu ra sẽ bị tắt cho luồng đó.

Writers

Mỗi bộ lọc có một người viết. Người viết xác định định dạng đầu ra thực tế cho thông tin đang được ghi. Trình ghi mặc định là trình ghi ASCII, tạo ra các tệp ASCII được phân tách bằng tab. Các tác giả khác có sẵn, như cho đầu ra nhị phân hoặc đăng nhập trực tiếp vào cơ sở dữ liệu.

Có một số cách khác nhau để tùy chỉnh ghi nhật ký của Zeek: bạn có thể tạo một dòng nhật ký mới, bạn có thể mở rộng nhật ký hiện có với các trường mới, bạn có thể áp dụng bộ lọc cho một dòng nhật ký hiện có hoặc bạn có thể tùy chỉnh định dạng đầu ra bằng cách đặt các tùy chọn ghi nhật ký . Tất cả các cách tiếp cận này được mô tả trong tài liệu này.

- ## Streams

Để ghi dữ liệu vào một luồng nhật ký mới, bạn cần thực hiện tất cả những việc sau:

- Một **record** kiểu phải được định nghĩa bao gồm tất cả các trường sẽ được ghi (theo quy ước, tên của loại bản ghi này thường là "Thông tin").
- ID luồng nhật ký (**enum** có tên loại **Log::ID**) phải được xác định để xác định duy nhất luồng nhật ký mới.
- Một dòng nhật ký phải được tạo bằng cách sử dụng **Log::create_stream** hàm.
- Khi dữ liệu được ghi có sẵn, **Log::write** hàm phải được gọi.

Trong ví dụ sau, chúng tôi tạo một mô-đun mới, mô-đun Foo này tạo một luồng nhật ký mới

```
module Foo;

export {
    # Create an ID for our new stream. By convention, this is
    # called "LOG".
    redef enum Log::ID += { LOG };

    # Define the record type that will contain the data to log.
    type Info: record {
        ts: time        &log;
        id: conn_id     &log;
        service: string &log &optional;
        missed_bytes: count &log &default=0;
    };
}

# Optionally, we can add a new field to the connection record so that
# the data we are logging (our "Info" record) will be easily
# accessible in a variety of event handlers.
redef record connection += {
    # By convention, the name of this new field is the lowercase name
    # of the module.
    foo: Info &optional;
};

# This event is handled at a priority higher than zero so that if
# users modify this stream in another script, they can do so at the
# default priority of zero.
event zeek_init() &priority=5
    {
    # Create the stream. This adds a default filter automatically.
    Log::create_stream(Foo::LOG, [$columns=Info, $path="foo"]);
    }
```

Trong định nghĩa của `Info` bản ghi ở trên, hãy chú ý rằng mỗi trường có **`&log`** thuộc tính. Nếu không có thuộc tính này, một trường sẽ không xuất hiện trong đầu ra nhật ký. Cũng lưu ý một trường có **`&optional`** thuộc tính. Điều này chỉ ra rằng trường có thể không được gán bất kỳ giá trị nào trước khi bản ghi nhật ký được ghi. Cuối cùng, một trường có **`&default`** thuộc tính có giá trị mặc định được gán cho nó một cách tự động.

Tại thời điểm này, điều duy nhất còn thiếu là một lệnh gọi **`Log::write`** hàm để gửi dữ liệu đến khung ghi nhật ký. Trình xử lý sự kiện thực tế nơi điều này sẽ diễn ra sẽ phụ thuộc vào nơi dữ liệu của bạn có sẵn. Trong ví dụ

này, **connection_established** sự kiện cung cấp dữ liệu của chúng tôi và chúng tôi cũng lưu trữ bản sao dữ liệu đang được đăng nhập vào **connection** bản ghi:

```
event connection_established(c: connection)
    {
    local rec: Foo::Info = [$ts=network_time(), $id=c$id];

    # Store a copy of the data in the connection record so other
    # event handlers can access it.
    c$foo = rec;

    Log::write(Foo::LOG, rec);
    }
```

Nếu bạn chạy Zeek với tập lệnh này, một tệp nhật ký mới foo.log sẽ được tạo. Mặc dù chúng tôi chỉ chỉ định bốn trường trong Info bản ghi ở trên, đầu ra nhật ký thực sự sẽ chứa bảy trường vì một trong các trường (trường được đặt tên id) chính là một loại bản ghi. Vì một **conn_id** bản ghi có bốn trường, nên mỗi trường này là một cột riêng biệt trong đầu ra nhật ký. Lưu ý rằng cách đặt tên các trường như vậy trong đầu ra nhật ký hơi khác với cách chúng ta đề cập đến cùng một trường trong tập lệnh Zeek (mỗi ký hiệu đô la được thay thế bằng dấu chấm). Ví dụ: để truy cập trường đầu tiên của a **conn_id** trong tập lệnh Zeek, chúng tôi sẽ sử dụng ký hiệu id$orig_h, nhưng trường đó được đặt tên id.orig_h trong đầu ra nhật ký.

Khi bạn đang phát triển các tập lệnh thêm dữ liệu vào **connection** bản ghi, bạn phải quan tâm đến thời gian và thời gian lưu trữ dữ liệu. Thông thường, dữ liệu được lưu vào bản ghi kết nối sẽ vẫn ở đó trong suốt thời gian kết nối và từ góc độ thực tế, không có gì lạ khi cần xóa dữ liệu đó trước khi kết thúc kết nối.

Thêm trường vào nhật ký

Bạn có thể thêm các trường bổ sung vào nhật ký bằng cách mở rộng loại bản ghi xác định nội dung của nó và đặt giá trị cho các trường mới trước khi mỗi bản ghi nhật ký được ghi.

Giả sử chúng tôi muốn thêm một trường boolean `is_private` để **`Conn::Info`** cho biết liệu địa chỉ IP của người khởi tạo có phải là một phần của**RFC 1918** không gian:

```
# Add a field to the connection log record.
redef record Conn::Info += {
    ## Indicate if the originator of the connection is part of the
    ## "private" address space defined in RFC1918.
    is_private: bool &default=F &log;
};
```

Như ví dụ này cho thấy, khi mở rộng bản ghi của luồng nhật ký `Info`, mỗi trường mới phải luôn được khai báo với giá trị & giá trị mặc định hoặc bằng **`&optional`**. Hơn nữa, bạn cần thêm **`&log`** thuộc tính nếu không trường sẽ không xuất hiện trong tệp nhật ký.

Bây giờ chúng ta cần thiết lập trường. Mặc dù các chi tiết khác nhau tùy thuộc vào nhật ký nào đang được mở rộng, nói chung điều quan trọng là phải chọn một sự kiện phù hợp để đặt các trường bổ sung vì chúng ta cần đảm bảo rằng các trường được đặt trước khi ghi bản ghi nhật ký. Đôi khi lựa chọn đúng là cùng một sự kiện ghi bản ghi nhật ký, nhưng ở mức độ ưu tiên cao hơn (để đảm bảo rằng trình xử lý sự kiện đặt các trường bổ sung được thực thi trước trình xử lý sự kiện ghi bản ghi nhật ký).

Trong ví dụ này, vì bản tóm tắt của kết nối được tạo tại thời điểm trạng thái của nó bị xóa khỏi bộ nhớ, chúng tôi có thể thêm một trình xử lý khác vào thời điểm đó để đặt trường của chúng tôi một cách chính xác:

```
event connection_state_remove(c: connection)
    {
    if ( c$id$orig_h in Site::private_address_space )
        c$conn$is_private = T;
    }
```

Bây giờ `conn.log` sẽ hiển thị một trường `is_private` loại mới **`bool`**. Nếu bạn nhìn vào tập lệnh Zeek xác định luồng bản ghi kết nối base / protocols / conn / main.zeek ,

bạn sẽ thấy điều đó **Log::write** được gọi trong một trình xử lý sự kiện cho cùng một sự kiện như được sử dụng trong ví dụ này để đặt các trường bổ sung, nhưng tại mức độ ưu tiên thấp hơn mức độ ưu tiên được sử dụng trong ví dụ này (tức là bản ghi nhật ký được ghi sau khi chúng ta gán `is_private` trường).

Để mở rộng nhật ký theo cách này, người ta cần một chút kiến thức về cách tập lệnh tạo dòng nhật ký đang tổ chức lưu giữ trạng thái của nó. Hầu hết các tập lệnh Zeek tiêu chuẩn đều đính kèm trạng thái nhật ký của chúng vào **connection** bản ghi mà sau đó nó có thể được truy cập, giống như `c$conn` ở trên. Ví dụ: phân tích HTTP thêm một trường kiểu http **HTTP::Info** vào **connection** bản ghi.

Xác định sự kiện ghi nhật ký

Đôi khi, việc phân tích bổ sung thông tin được ghi lại sẽ rất hữu ích. Đối với những trường hợp này, một luồng có thể chỉ định một sự kiện sẽ được tạo mỗi khi một bản ghi nhật ký được ghi vào đó. Để làm điều này, chúng tôi cần sửa đổi mô-đun ví dụ được hiển thị ở trên để trông giống như sau:

```
module Foo;

export {
    redef enum Log::ID += { LOG };

    type Info: record {
        ts: time     &log;
        id: conn_id  &log;
        service: string &log &optional;
        missed_bytes: count &log &default=0;
    };

    # Define a logging event. By convention, this is called
    # "log_<stream>".
    global log_foo: event(rec: Info);
}

event zeek_init() &priority=5
    {
    # Specify the "log_foo" event here in order for Zeek to raise it.
    Log::create_stream(Foo::LOG, [$columns=Info, $ev=log_foo,
                        $path="foo"]);
    }
```

Tất cả các luồng nhật ký mặc định của Zeek xác định một sự kiện như vậy. Ví dụ: dòng nhật ký kết nối nâng cao sự kiện **Conn::log_conn**. Bạn có thể sử dụng ví dụ đó để gắn cờ khi kết nối đến một điểm đến cụ thể vượt quá một thời lượng nhất định:

```
redef enum Notice::Type += {
    ## Indicates that a connection remained established longer
    ## than 5 minutes.
    Long_Conn_Found
};

event Conn::log_conn(rec: Conn::Info)
    {
    if ( rec?$duration && rec$duration > 5mins )
        NOTICE([$note=Long_Conn_Found,
                $msg=fmt("unusually long conn to %s", rec$id$resp_h),
                $id=rec$id]);
    }
```

Thông thường, những sự kiện này có thể là một giải pháp thay thế cho quá trình hậu xử lý nhật ký Zeek bên ngoài bằng các tập lệnh Perl. Phần lớn những gì mà một tập lệnh bên ngoài như vậy sẽ thực hiện ngoại tuyến sau này, thay vào đó, một tập lệnh bên ngoài có thể thực hiện trực tiếp bên trong Zeek trong thời gian thực.

Tắt luồng

Một cách để "tắt" nhật ký là tắt hoàn toàn luồng. Ví dụ: ví dụ sau sẽ ngăn conn.log được viết:

```
event zeek_init()
    {
    Log::disable_stream(Conn::LOG);
    }
```

Lưu ý rằng điều này phải chạy sau khi luồng được tạo, vì vậy mức độ ưu tiên của trình xử lý sự kiện này phải thấp hơn mức độ ưu tiên của trình xử lý sự kiện nơi luồng được tạo.

- **Filters**

Luồng có một hoặc nhiều bộ lọc được gắn vào luồng đó. Luồng không có bất kỳ bộ lọc nào sẽ không tạo ra bất kỳ đầu ra nhật ký nào. Các bộ lọc chi phối hai khía cạnh của sản xuất nhật ký: chúng kiểm soát mục nhập nhật ký nào của luồng được ghi ra ngoài và chúng xác định cách thực hiện việc ghi nhật ký. Họ thực hiện điều sau bằng cách chỉ định trình ghi nhật ký thực hiện thao tác ghi, chẳng hạn như trình ghi ASCII (xem bên dưới) cho đầu ra tệp văn bản. Khi một luồng được tạo, luồng đó sẽ tự động được gắn một bộ lọc mặc định vào. Bộ lọc mặc định này có thể bị xóa hoặc thay thế hoặc có thể thêm các bộ lọc khác vào luồng. Điều này được thực hiện bằng cách sử dụng **Log::add_filter** hoặc **Log::remove_filter** hàm số. Phần này chỉ ra cách sử dụng bộ lọc để thực hiện các tác vụ như đổi tên tệp nhật ký, chia đầu ra thành nhiều tệp, kiểm soát bản ghi nào được ghi và đặt khoảng thời gian xoay vòng tùy chỉnh.

Mỗi bộ lọc có một tên duy nhất, có phạm vi đến luồng mà bộ lọc đó chứa. Nghĩa là, tất cả các bộ lọc được gắn vào một luồng nhất định đều có tên khác nhau. Việc gọi **Log::add_filter** để thêm bộ lọc có tên đã tồn tại cho luồng sẽ thay thế bộ lọc hiện có.

Đổi tên tệp nhật ký

Thông thường, tên tệp nhật ký cho một luồng nhật ký nhất định được xác định khi tạo luồng, trừ khi bạn chỉ định rõ ràng một tên khác bằng cách thêm bộ lọc.

Cách dễ nhất để thay đổi tên tệp nhật ký là chỉ cần thay thế bộ lọc nhật ký mặc định bằng một bộ lọc mới chỉ định giá trị cho `path` trường. Trong ví dụ này, `conn.log` sẽ được thay đổi thành `myconn.log`:

```
event zeek_init()
    {
    # Replace default filter for the Conn::LOG stream in order to
    # change the log filename.

    local f = Log::get_filter(Conn::LOG, "default");
    f$path = "myconn";
    Log::add_filter(Conn::LOG, f);
    }
```

Hãy nhớ rằng `path` trường của bộ lọc nhật ký không bao giờ chứa phần mở rộng tên tệp. Phần mở rộng sẽ được xác định sau bởi người viết nhật ký.

Thêm tệp đầu ra bổ sung

Thông thường, một luồng nhật ký chỉ ghi vào một tệp nhật ký. Tuy nhiên, bạn có thể thêm bộ lọc để luồng ghi vào nhiều tệp. Điều này rất hữu ích nếu bạn muốn hạn chế tập hợp các trường được ghi vào tệp mới.

Trong ví dụ này, một bộ lọc mới được thêm vào **Conn::LOG** luồng ghi hai trường vào tệp nhật ký mới:

```
event zeek_init()
    {
    # Add a new filter to the Conn::LOG stream that logs only
    # timestamp and originator address.

    local filter: Log::Filter = [$name="orig-only", $path="origs",
                                 $include=set("ts", "id.orig_h")];
    Log::add_filter(Conn::LOG, filter);
    }
```

Lưu ý cách `include` thuộc tính bộ lọc chỉ định một tập hợp giới hạn các trường đối với những trường đã cho. Các tên tương ứng với các tên trong **Conn::Info** bản ghi (tuy nhiên, vì `id` bản thân trường là một bản ghi, chúng ta có thể chỉ định một trường riêng lẻ `id` bằng ký hiệu dấu chấm được hiển thị trong ví dụ).

Sử dụng mã trên, ngoài tệp thông thường `conn.log`, giờ đây bạn cũng sẽ nhận được tệp nhật ký mới `origs.log` trông giống như tệp thông thường `conn.log`, nhưng sẽ chỉ có các trường được chỉ định trong `include` thuộc tính bộ lọc.

Nếu bạn chỉ muốn bỏ qua một số trường nhưng giữ phần còn lại, có một thuộc tính bộ lọc loại trừ tương ứng mà bạn có thể sử dụng thay vì bao gồm để chỉ liệt kê những trường mà bạn không quan tâm.

Nếu bạn muốn đặt đây là tệp nhật ký duy nhất cho luồng, bạn có thể xóa bộ lọc mặc định:

```
event zeek_init()
    {
    # Remove the filter called "default".
    Log::remove_filter(Conn::LOG, "default");
    }
```

Xác định động đường dẫn nhật ký

Thay vì sử dụng `path` thuộc tính bộ lọc, bộ lọc có thể xác định *động* các đường dẫn đầu ra dựa trên bản ghi đang được ghi. Điều đó cho phép, ví dụ, ghi lại các kết nối cục bộ và từ xa thành các tệp riêng biệt. Để làm điều này, bạn xác định một hàm trả về đường dẫn mong muốn và sử dụng `path_func` thuộc tính bộ lọc:

```
# Note: if using ZeekControl then you don't need to redef local_nets.
redef Site::local_nets = { 192.168.0.0/16 };

function myfunc(id: Log::ID, path: string, rec: Conn::Info) : string
    {
    # Return "conn-local" if originator is a local IP, otherwise
    # return "conn-remote".
    local r = Site::is_local_addr(rec$id$orig_h) ? "local" : "remote";
    return fmt("%s-%s", path, r);
    }

event zeek_init()
    {
    local filter: Log::Filter = [$name="conn-split",
             $path_func=myfunc, $include=set("ts", "id.orig_h")];
    Log::add_filter(Conn::LOG, filter);
    }
```

Chạy nó bây giờ sẽ tạo ra hai tệp mới `conn-local.log` và `conn-remote.log`, với các mục nhập tương ứng (để ví dụ này hoạt động, **`Site::local_nets`** phải chỉ định mạng cục bộ của bạn). Người ta có thể mở rộng điều này hơn nữa, ví dụ để ghi thông tin bằng mạng con hoặc thậm chí bằng địa chỉ IP. Tuy nhiên, hãy cẩn thận vì có thể dễ dàng tạo ra nhiều tệp rất nhanh chóng.

Hàm `myfunc` có một nhược điểm: nó chỉ có thể được sử dụng với **`Conn::LOG`** luồng vì kiểu bản ghi được mã hóa cứng vào danh sách đối số của nó. Tuy nhiên, Zeek cho phép thực hiện một biến thể chung chung hơn:

```
function myfunc(id: Log::ID, path: string,
                rec: record { id: conn_id; } ) : string
    {
    local r = Site::is_local_addr(rec$id$orig_h) ? "local" : "remote";
    return fmt("%s-%s", path, r);
    }
```

Chức năng này có thể được sử dụng với tất cả các luồng nhật ký có các bản ghi chứa một trường. `id: conn_id`

Lọc bản ghi nhật ký

Chúng tôi vừa xem các cách tùy chỉnh các cột đã ghi. Khung ghi nhật ký cũng cho phép bạn kiểm soát bản ghi nào mà Zeek ghi ra. Nó dựa vào cơ chế của Zeek **`hook`** để làm điều này, như sau. Khung cung cấp hai cấp hook "chính sách", một cấp tổng thể và một tập hợp các cấp bộ lọc. Trình xử lý hook có thể thực hiện xử lý bổ sung một bản ghi nhật ký, bao gồm cả việc phủ quyết việc ghi bản ghi. Bất kỳ trình xử lý nào sử dụng một **`break`** câu lệnh để rời khỏi hook đều tuyên bố rằng một bản ghi sẽ không được ghi ra ngoài. Bất kỳ ai cũng có thể gắn trình xử lý vào các hook này, trông như sau:

```
type Log::StreamPolicyHook: hook(rec: any, id: ID);
type Log::PolicyHook: hook(rec: any, id: ID, filter: Filter);
```

Đối với cả hai loại hook, rec đối số chứa mục nhập được ghi nhật ký và là một thể hiện của loại bản ghi được liên kết với các cột của luồng và id xác định luồng nhật ký.

Khung ghi nhật ký xác định một hook chính sách hook chung **Log::log_stream_policy**:. Đối với mỗi lần ghi nhật ký, hook này sẽ được gọi đầu tiên. Bất kỳ trình xử lý nào của nó đều có thể quyết định phủ quyết mục nhập nhật ký. Sau đó, khuôn khổ sẽ lặp lại các bộ lọc của luồng nhật ký. Mỗi bộ lọc có một filter$policy hook loại **Log::PolicyHook**. Các trình xử lý của nó nhận bản ghi nhật ký, ID của luồng nhật ký và chính bản ghi bộ lọc. Mỗi trình xử lý có thể phủ quyết việc viết. Sau khi hook của bộ lọc đã chạy, bất kỳ quyền phủ quyết nào (bởi **Log::log_stream_policy** hoặc hook của bộ lọc) sẽ hủy bỏ việc ghi thông qua bộ lọc đó. Nếu không có quyền phủ quyết nào xảy ra, bộ lọc hiện hướng bản ghi nhật ký đến đầu ra của nó.

Bạn có thể chuyển trạng thái tùy ý thông qua các trình xử lý hook này. Ví dụ: bạn có thể mở rộng luồng hoặc bộ lọc qua a **redef** hoặc chuyển các cặp khóa-giá trị qua filter$config bảng ..

Vì bạn thường muốn sử dụng xử lý thống nhất cho tất cả các lần ghi trên một luồng nhất định, các luồng nhật ký cung cấp một hook mặc định, được cung cấp khi xây dựng luồng, mà các bộ lọc của luồng sẽ sử dụng nếu chúng không cung cấp các luồng riêng. Để hỗ trợ hook trên luồng nhật ký của bạn, bạn phải luôn xác định hook mặc định khi tạo luồng mới, như sau:

```
module Foo;

export {
    ## The logging stream identifier.
    redef enum Log::ID += { LOG };

    ## A default logging policy hook for the stream.
    global log_policy: Log::PolicyHook;

    # Define the record type that will contain the data to log.
    type Info: record {
        ts: time        &log;
        id: conn_id     &log;
        service: string &log &optional;
        missed_bytes: count &log &default=0;
    };
}

event zeek_init() &priority=5
    {
    # Create the stream, adding the default policy hook:
    Log::create_stream(Foo::LOG, [$columns=Info, $path="foo", $policy=log_policy]);
    }
```

Với hook này tại chỗ, bây giờ thật dễ dàng để thêm một vị từ lọc cho `Foo` nhật ký từ mọi nơi:

```
hook Foo::log_policy(rec: Foo::Info, id: Log::ID, filter: Log::Filter)
    {
    # Let's only log complete information:
    if ( rec$missed_bytes > 0 )
        break;
    }
```

Bản phân phối Zeek có các hook mặc định cho tất cả các luồng của nó. Đây là một ví dụ thực tế hơn, sử dụng HTTP:

```
hook HTTP::log_policy(rec: HTTP::Info, id: Log::ID, filter: Log::Filter)
    {
    # Record only connections with successfully analyzed HTTP traffic
    if ( ! rec?$service || rec$service != "http" )
        break;
    }
```

Để ghi đè hook một cách chọn lọc trong bộ lọc mới, hãy đặt hook khi thêm bộ lọc vào luồng:

```
hook my_policy(rec: Foo::Info, id: Log::ID, filter: Log::Filter)
    {
    # Let's only log incomplete flows:
    if ( rec$missed_bytes == 0 )
        break;
    }

event zeek_init()
    {
    local filter: Log::Filter = [$name="incomplete-only",
                                 $path="foo-incomplete",
                                 $policy=my_policy];
    Log::add_filter(Foo::LOG, filter);
    }
```

Lưu ý rằng cách tiếp cận này có ý nghĩa tinh tế: bộ lọc mới không sử dụng `Foo::log_policy` hook và hook đó không được gọi để ghi vào bộ lọc này. Mọi quyền phủ quyết hoặc xử lý bổ sung được triển khai trong `Foo::log_policy` trình xử lý sẽ không còn xảy ra đối với bộ lọc mới. Việc thay thế hook như vậy hiếm khi cần thiết; bạn có thể thấy tốt hơn nếu thu hẹp trình xử lý mặc định của luồng thành bộ lọc được đề cập:

```
hook Foo::log_policy(rec: Foo::Info, id: Log::ID, filter: Log::Filter)
    {
    if ( filter$name != "incomplete-only" )
        return;

    # Let's only log incomplete flows:
    if ( rec$missed_bytes == 0 )
        break;
    }
```

Đối với các tác vụ cần chạy một lần cho mỗi lần ghi, không phải một lần cho mỗi lần ghi và bộ lọc, hãy sử dụng **`Log::log_stream_policy`**:

```
hook Log::log_stream_policy(rec: Foo::Info, id: Log::ID)
    {
    # Called once per write
    }

hook Foo::log_policy(rec: Foo::Info, id: Log::ID, filter: Log::Filter)
    {
    # Called once for each of Foo's filters.
    }
```

Để thay đổi một bộ lọc hiện có, trước tiên hãy truy xuất nó, sau đó cập nhật nó và thiết lập lại nó:

```
hook my_policy(rec: Foo::Info, id: Log::ID, filter: Log::Filter)
    {
    # Let's only log incomplete flows:
    if ( rec$missed_bytes == 0 )
        break;
    }

event zeek_init()
    {
    local f = Log::get_filter(Foo::LOG, "default");
    f$policy = my_policy;
    Log::add_filter(Foo::LOG, f);
    }
```

Xoay nhật ký và xử lý hậu kỳ

Khung ghi nhật ký cung cấp khả năng kiểm soát chi tiết về thời điểm và cách xoay các tệp nhật ký. Xoay vòng nhật ký có nghĩa là Zeek định kỳ đổi tên tệp nhật ký đang hoạt động, chẳng hạn như `conn.log`, theo cách mà người dùng có thể định cấu hình (ví dụ: đổi tên thành để `conn_21-01-03_14-05-00.log` đánh dấu thời gian) và bắt đầu lại trên một `conn.log` tệp mới. Xử lý hậu kỳ có nghĩa là Zeek cũng có thể áp dụng xử lý bổ sung tùy chọn cho tệp được xoay, chẳng hạn như nén hoặc truyền tệp. Các cơ chế này áp dụng tự nhiên cho người viết nhật ký dựa trên tệp, nhưng cũng có sẵn cho những người viết khác cũng như các dạng tổng quát hơn của quá trình xử lý bổ sung định kỳ đối với kết quả đầu ra của họ.

Thời gian xoay vòng

Khoảng thời gian xoay vòng nhật ký có thể được kiểm soát toàn cầu cho tất cả các bộ lọc bằng cách xác định lại **Log::default_rotation_interval** hằng số hoặc cụ thể cho các **Log::Filter** trường hợp nhất định bằng cách thiết lập interv trường của chúng. Giá trị mặc định 0secs, vô hiệu hóa xoay.

Dưới đây là một ví dụ về việc chỉ thay đổi **Conn::LOG** xoay vòng bộ lọc mặc định của luồng:

```
event zeek_init()
    {
    local f = Log::get_filter(Conn::LOG, "default");
    f$interv = 1 min;
    Log::add_filter(Conn::LOG, f);
    }
```

Kiểm soát đặt tên tệp

Redf'able **Log::rotation_format_func** xác định tên của tệp được xoay. Khung ghi nhật ký gọi hàm với đủ ngữ cảnh (một **Log::RotationFmtInfo** bản ghi), từ đó nó xác định tên đầu ra trong hai phần: thư mục đầu ra và tên cơ sở của tệp đầu ra, nghĩa là tên của nó không có hậu tố. Nó trả về hai thành phần này thông qua một **Log::RotationPath** bản ghi. Thư mục đầu ra mặc định là **Log::default_rotation_dir** (một tùy chọn cấu hình) và kết hợp một dấu thời gian trong tên cơ sở, như được chỉ định bởi **Log::default_rotation_date_format**.

Xử lý sau các bản ghi đã xoay

Quá trình xử lý hậu kỳ có thể được tiến hành thông qua các mặc định được định cấu hình trên tất cả các bộ lọc nhật ký hoặc với các tùy chỉnh cho mỗi bộ lọc. Zeek cung cấp cơ sở hạ tầng mặc định hữu ích để đơn giản hóa việc chạy các lệnh shell trên các bản ghi đã xoay, nhưng bạn có thể tự do xác định cơ sở hạ tầng xử lý sau của riêng mình từ đầu.

Theo mặc định **Log::default_rotation_postprocessor_cmd**,, nếu được xác định, chạy trên mọi nhật ký được xoay. Hàm wrapper thực hiện lệnh gọi thực sự

là **Log::run_rotation_postprocessor_cmd**. Nó chuyển sáu đối số bổ sung cho lệnh shell đã định cấu hình:

- Tên tệp được xoay thành (ví dụ `conn_21-01-03_14-05-00.log`:)
- Tên cơ sở ban đầu (ví dụ `conn`)
- Dấu thời gian mà tại đó tệp nhật ký ban đầu được tạo (ví dụ `21-01-03_14.04.00`:)
- Dấu thời gian mà tại đó tệp nhật ký ban đầu được xoay (ví dụ `21-01-03_15.05.00`:)
- `1` nếu Zeek chấm dứt, `0` nếu không
- Tên của người viết (ví dụ: `ascii` đối với người viết ASCII)

Zeek cung cấp các bộ xử lý hậu kỳ sẵn sàng sử dụng để truyền tệp qua SCP và SFTP . Dự án Zeek cũng cung cấp một công cụ bên ngoài, zeek-archiver , thực hiện nén nhật ký bên ngoài quy trình Zeek để tạo sự mạnh mẽ.

Các tính năng khác

- Ánh xạ tên trường

Đôi khi, có thể hữu ích khi viết lại tên cột khi chúng xuất hiện trong nhật ký Zeek. Trường hợp sử dụng điển hình cho điều này sẽ là đảm bảo rằng việc đặt tên cột tuân thủ các yêu cầu của hệ thống nhập nhật ký của bạn. Để đạt được điều này, bạn có thể cung cấp bản đồ dịch tên và tại đây, bạn cũng có thể thực hiện việc này trên toàn cầu hoặc theo từng bộ lọc. Các bản đồ là các bảng chuỗi đơn giản với các khóa là tên trường của Zeek và các giá trị là những giá trị thực sự cần ghi ra. Các tên trường không có trong bản đồ vẫn không thay đổi. Biến thể toàn cục là (thường trống) **Log::default_field_name_map** và tương đương bộ lọc cục bộ tương ứng là `field_name_map` thành viên của bộ lọc.

Ví dụ: bản đồ tên sau đây loại bỏ các dấu chấm trong cách đặt tên thông thường của các ID kết nối:

```
redef Log::default_field_name_map = {
     ["id.orig_h"] = "id_orig_h",
     ["id.orig_p"] = "id_orig_p",
     ["id.resp_h"] = "id_resp_h",
     ["id.resp_p"] = "id_resp_p"
};
```

Tất cả nhật ký hiển thị một bộ mã định danh kết nối hiện sử dụng…

```
#fields  ts  uid  id_orig_h  id_orig_p  id_resp_h  id_resp_p ...
```

Thay vì các tên mặc định:

```
#fields  ts  uid  id.orig_h  id.orig_p  id.resp_h  id.resp_p ...
```

Nếu bạn chỉ muốn thay đổi này cho một bộ lọc nhật ký nhất định, hãy thực hiện thay đổi trực tiếp đối với bản ghi bộ lọc. Những điều sau đây chỉ thay đổi cách đặt tên cho `conn.log`:

```
event zeek_init()
   {
   local f = Log::get_filter(Conn::LOG, "default");
   f$field_name_map = table(
       ["id.orig_h"] = "id_orig_h",
       ["id.orig_p"] = "id_orig_p",
       ["id.resp_h"] = "id_resp_h",
       ["id.resp_p"] = "id_resp_p");
   Log::add_filter(Conn::LOG, f);
   }
```

- In để ghi nhật ký tin nhắn

Câu lệnh của Zeek **print** thường ghi vào stdout hoặc một tệp đầu ra cụ thể. Bằng cách điều chỉnh **Log::print_to_log** giá trị enum, bạn có thể chuyển hướng các câu lệnh như vậy để chuyển trực tiếp vào nhật ký Zeek. Các giá trị có thể bao gồm:

- **Log::REDIRECT_NONE** : mặc định, không liên quan đến nhật ký Zeek
- **Log::REDIRECT_STDOUT** : các bản in thường chuyển đến stdout đi đến nhật ký
- **Log::REDIRECT_ALL** : bất kỳ bản in nào kết thúc trong nhật ký thay vì stdout hoặc các tệp khác

Định **Log::print_log_path** nghĩa tên của tệp nhật ký, **Log::PrintLogInfo** các cột và **Log::log_print** sự kiện của nó cho phép bạn xử lý các thông báo đã ghi nhật ký thông qua các trình xử lý sự kiện.

- Ghi nhật ký cục bộ và từ xa

Trong quá trình xử lý nhật ký của mình, Zeek xem xét liệu việc ghi nhật ký có xảy ra cục bộ đối với một nút Zeek hay từ xa trên một nút khác, sau khi chuyển tiếp các mục nhật ký tới nó. Thiết lập Zeek nút đơn mặc định là ghi nhật ký cục bộ, trong khi thiết lập cụm chỉ cho phép ghi nhật ký cục bộ trên các nút trình ghi nhật ký và ghi nhật ký từ xa trên tất cả trừ các nút trình ghi nhật ký. Thông thường, bạn không cần phải đến gần các cài đặt này, nhưng bạn có thể làm như vậy bằng cách redef 'ing the **Log::enable_local_logging** và **Log::enable_remote_logging** booleans, tương ứng.

- **Writers**

Mỗi bộ lọc có một người viết. Nếu bạn không chỉ định trình viết khi thêm bộ lọc vào luồng, thì trình ghi ASCII là mặc định.

Có hai cách để chỉ định người viết không phải mặc định. Để thay đổi trình ghi mặc định cho tất cả các bộ lọc nhật ký, chỉ cần xác định lại **Log::default_writer** tùy chọn. Ngoài ra, bạn có thể chỉ định người viết sử dụng trên cơ sở từng bộ lọc bằng cách đặt giá trị cho trường của bộ lọc writer. Tham khảo tài liệu của người viết để sử dụng để xem có các tùy chọn khác cần thiết hay không.

ASCII Writers

Theo mặc định, trình ghi ASCII xuất ra các tệp nhật ký bắt đầu bằng một số dòng siêu dữ liệu, sau đó là đầu ra nhật ký thực tế. Siêu dữ liệu mô tả định dạng của tệp

nhật ký, path của nhật ký (tức là tên tệp nhật ký không có phần mở rộng tệp) và cũng chỉ định thời gian nhật ký được tạo và thời điểm Zeek ghi xong vào đó. Trình viết ASCII có một số tùy chọn để tùy chỉnh định dạng đầu ra của nó, hãy xem cơ sở / khung / ghi nhật ký / nhà văn / ascii.zeek . Nếu bạn thay đổi các tùy chọn định dạng đầu ra, hãy cẩn thận kiểm tra xem các tập lệnh xử lý sau của bạn có còn có thể nhận ra các tệp nhật ký của bạn hay không.

Một số tùy chọn người viết là chung (nghĩa là chúng ảnh hưởng đến tất cả các bộ lọc nhật ký sử dụng người viết nhật ký đó). Ví dụ: để thay đổi định dạng đầu ra của tất cả nhật ký ASCII thành định dạng JSON:

```
redef LogAscii::use_json = T;
```

Một tùy chọn chung tương tự là tùy chọn logdir chỉ định một thư mục làm vị trí cho các tệp đầu ra.

```
redef LogAscii::logdir = output_directory;
```

Cả hai đều có thể được sử dụng từ dòng lệnh, một mình hoặc cùng với các tập lệnh khác:

```
zeek -r ../test-capture.cap LogAscii::use_json=T

mkdir output_directory ; zeek -r ../test-capture.cap LogAscii::logdir=output_directory
```

Một số tùy chọn người viết là bộ lọc cụ thể (nghĩa là chúng chỉ ảnh hưởng đến các bộ lọc chỉ định rõ ràng tùy chọn). Ví dụ, để thay đổi định dạng đầu ra của conn.log chỉ:

```
event zeek_init()
    {
    local f = Log::get_filter(Conn::LOG, "default");
    # Use tab-separated-value mode
    f$config = table(["tsv"] = "T");
    Log::add_filter(Conn::LOG, f);
    }
```

SQLite Writer

SQLite là một hệ thống cơ sở dữ liệu SQL đơn giản, dựa trên tệp, được sử dụng rộng rãi. Sử dụng SQLite cho phép Zeek ghi và truy cập dữ liệu ở định dạng dễ sử dụng để trao đổi với các ứng dụng khác. Do tính chất giao dịch của SQLite, cơ sở dữ liệu có thể được một số ứng dụng sử dụng đồng thời. Khung nhập liệu của Zeek hỗ trợ trình đọc SQLite .

Hỗ trợ ghi nhật ký cho SQLite có sẵn trong tất cả các cài đặt Zeek. Không cần tải bất kỳ tập lệnh bổ sung nào hoặc cho bất kỳ cấu hình thời gian biên dịch nào. Gửi dữ liệu từ các luồng ghi nhật ký hiện có tới SQLite khá đơn giản. Nhiều khả năng bạn sẽ chỉ muốn đầu ra SQLite cho các bộ lọc nhật ký được chọn, vì vậy bạn phải định cấu hình một bộ lọc để sử dụng trình viết SQLite. Mã ví dụ sau thêm SQLite làm bộ lọc cho nhật ký kết nối:

```
event zeek_init()
    {
    local filter: Log::Filter =
        [
        $name="sqlite",
        $path="/var/db/conn",
        $config=table(["tablename"] = "conn"),
        $writer=Log::WRITER_SQLITE
        ];

     Log::add_filter(Conn::LOG, filter);
    }
```

Zeek sẽ tạo tệp cơ sở dữ liệu `/var/db/conn.sqlite` nếu nó chưa tồn tại. Nó cũng sẽ tạo một bảng với tên `conn` (nếu nó không tồn tại) và bắt đầu thêm thông tin kết nối vào bảng.

Zeek hiện không hỗ trợ cơ sở dữ liệu SQLite luân phiên như đối với nhật ký ASCII. Bạn phải chăm sóc để tạo chúng ở những vị trí thích hợp.

Nếu bạn kiểm tra cơ sở dữ liệu SQLite kết quả, lược đồ sẽ chứa các trường giống nhau có trong tệp nhật ký ASCII:

```
sqlite3 /var/db/conn.sqlite
```

```
SQLite version 3.8.0.2 2013-09-03 17:11:13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE conn (
'ts' double precision,
'uid' text,
'id.orig_h' text,
'id.orig_p' integer,
...
```

Lưu ý rằng với mã trên, ASCII `conn.log` sẽ vẫn được tạo, vì nó thêm một bộ lọc nhật ký bổ sung cùng với bộ ghi nhật ký ASCII mặc định. Để tránh điều này, bạn có thể xóa bộ lọc mặc định:

```
Log::remove_filter(Conn::LOG, "default");
```

Để tạo tệp nhật ký SQLite tùy chỉnh, bạn phải tạo một dòng nhật ký mới chỉ chứa thông tin bạn muốn cam kết với cơ sở dữ liệu.

❖ ***Framework Thông báo***

Một trong những cách dễ nhất để tùy chỉnh Zeek là viết chính sách thông báo cục bộ. Zeek có thể phát hiện một số lượng lớn các tình huống thú vị tiềm ẩn và móc chính sách thông báo xác định chúng mà người dùng muốn được xử lý theo một cách nào đó. Đặc biệt, chính sách thông báo có thể chỉ định các hành động cần thực hiện, chẳng hạn như gửi email hoặc soạn email báo động thông thường.

- **Tổng quan**

Nhắc qua một chút kiến thức cơ bản về triết lý của Zeek về việc báo cáo mọi thứ. Zeek cung cấp một số lượng lớn các tập lệnh chính sách thực hiện nhiều loại phân tích. Hầu hết các tập lệnh này giám sát hoạt động mà người dùng có thể quan tâm. Tuy nhiên, không có tập lệnh nào trong số này xác định tầm quan trọng của những gì nó tự tìm thấy. Thay vào đó, các tập lệnh chỉ gắn cờ các tình huống *có khả năng* thú vị, để nó cho cấu hình cục bộ để xác định tình huống nào trong số chúng thực tế có thể hành động được. Việc phân tách phát hiện và báo cáo này cho phép Zeek giải quyết các nhu cầu khác nhau mà các trang web khác nhau có. Định nghĩa về những gì cấu thành một cuộc tấn công hoặc thậm chí một sự thỏa hiệp khác nhau khá nhiều giữa các môi trường và hoạt động được coi là độc hại tại một trang web có thể hoàn toàn được chấp nhận ở một trang web khác.

Bất cứ khi nào một trong các tập lệnh phân tích của Zeek thấy điều gì đó thú vị, nó sẽ gắn cờ tình huống bằng cách gọi **NOTICE** hàm và tạo cho nó một **Notice::Info** bản ghi duy nhất. Thông báo có một **Notice::Type**, phản ánh loại hoạt động đã được nhìn thấy, và nó cũng thường được bổ sung thêm bối cảnh về tình huống. Có thể tìm thêm thông tin về việc nâng cao thông báo trong phần Nâng cao thông báo .

Sau khi một thông báo được đưa ra, nó có thể có bất kỳ hành động nào được áp dụng cho nó bằng cách viết các **Notice::policy** hook được mô tả trong phần Chính sách Thông báo bên dưới. Những hành động như vậy có thể ví dụ như để gửi thư đến (các) địa chỉ đã định cấu hình hoặc đơn giản là bỏ qua thông báo. Hiện tại, các hành động sau được xác định:

Hoạt động	Sự mô tả
Notice::ACTION_LOG	Viết thông báo vào **Notice::LOG** luồng ghi nhật ký.
Notice::ACTION_ALARM	Đăng nhập vào **Notice::ALARM_LOG** luồng sẽ xoay vòng hàng giờ và gửi nội dung qua email đến địa chỉ email hoặc các địa chỉ trong

Hoạt động	Sự mô tả
	trường *email_dest* của **Notice::Info** bản ghi thông báo đó.
Notice::ACTION_EMAIL	Gửi thông báo bằng email đến địa chỉ email hoặc các địa chỉ trong trường *email_dest* của **Notice::Info** hồ sơ thông báo đó.
Notice::ACTION_PAGE	Gửi email đến địa chỉ email hoặc các địa chỉ trong trường *email_dest* của **Notice::Info** hồ sơ thông báo đó.

Cách các hành động thông báo này được áp dụng cho các thông báo được thảo luận trong phần Chính sách Thông báo và Các phím tắt của Chính sách Thông báo .

- **Thông báo xử lý**

Chính sách Thông báo

Hook **Notice::policy** cung cấp cơ chế để áp dụng các hành động và thường sửa đổi thông báo trước khi nó được gửi tới các plugin hành động. Hook có thể được coi là các chức năng đa thân và việc sử dụng chúng trông rất giống với việc xử lý các sự kiện. Sự khác biệt là chúng không đi qua hàng đợi sự kiện như các sự kiện. Người dùng có thể thay đổi quá trình xử lý thông báo bằng cách sửa đổi trực tiếp các trường trong **Notice::Info** bản ghi được cung cấp làm đối số cho hook.

Đây là một ví dụ đơn giản yêu cầu Zeek gửi email cho tất cả các loại thông báo **SSH::Password_Guessing** nếu người đoán cố gắng đăng nhập vào máy chủ tại 192.168.56.103 :

```
@load protocols/ssh/detect-bruteforcing

redef SSH::password_guesses_limit=10;

hook Notice::policy(n: Notice::Info)
    {
    if ( n$note == SSH::Password_Guessing && /192\.168\.56\.103/ in n$sub )
        {
        add n$actions[Notice::ACTION_EMAIL];
        n$email_dest = "ssh_alerts@example.net";
        }
    }
```

```
$ zeek -C -r ssh/sshguess.pcap notice_ssh_guesser.zeek
$ cat notice.log
```

```
#separator \x09
#set_separator    ,
#empty_field      (empty)
#unset_field      -
#path     notice
#open     2018-12-13-22-56-35
#fields   ts      uid     id.orig_h       id.orig_p       id.resp_h       id.resp_p       fuid    fil
#types    time    string  addr    port    addr    port    string  string  string  enum    enum    str
1427726759.303199 -       -       -       -       -       -       -       -       -       SSH::Passwo
#close    2018-12-13-22-56-35
```

Các hook cũng có thể có các mức độ ưu tiên được áp dụng để sắp xếp việc thực thi của chúng như các sự kiện có mức độ ưu tiên mặc định là 0. Giá trị lớn hơn được thực thi trước. Đặt nội dung hook để chạy trước khi các nội dung hook mặc định có thể trông như thế này:

```
hook Notice::policy(n: Notice::Info) &priority=5
    {
    # Insert your code here.
    }
```

Các phím tắt chính sách thông báo

Mặc dù khung thông báo cung cấp rất nhiều tính linh hoạt và khả năng cấu hình, nhưng nhiều khi không cần đến sự thể hiện đầy đủ và thực sự trở thành một trở ngại

cho việc đạt được kết quả. Khung công tác cung cấp một nội dung **Notice::policy** hook mặc định như một cách cung cấp cho người dùng các phím tắt để dễ dàng áp dụng nhiều hành động phổ biến cho các thông báo.

Chúng được triển khai dưới dạng các tập hợp và bảng được lập chỉ mục với một **Notice::Type** giá trị enum. Sau đây hiển thị và mô tả tất cả các biến có sẵn cho cấu hình phím tắt của khung thông báo.

- **Notice::ignored_types**

Thêm một **Notice::Type** vào tập hợp này dẫn đến thông báo bị bỏ qua. Nó sẽ không có bất kỳ hành động nào khác được áp dụng cho nó, thậm chí không **Notice::ACTION_LOG**.

- **Notice::emailed_types**

Thêm một **Notice::Type** vào tập hợp này dẫn đến **Notice::ACTION_EMAIL** việc được áp dụng cho các thông báo thuộc loại đó.

- **Notice::alarmed_types**

Thêm một **Notice::Type** vào tập hợp này dẫn đến **Notice::ACTION_ALARM** việc được áp dụng cho các thông báo thuộc loại đó.

- **Notice::not_suppressed_types**

Thêm một **Notice::Type** vào tập hợp này dẫn đến thông báo đó không còn trải qua quá trình chặn thông báo bình thường mà sẽ diễn ra. Hãy cẩn thận khi sử dụng điều này trong sản xuất, nó có thể làm tăng đáng kể số lượng thông báo được xử lý.

- **Notice::type_suppression_intervals**

Đây là một bảng được lập chỉ mục **Notice::Type** và cho ra một khoảng thời gian. Nó có thể được sử dụng như một cách dễ dàng để kéo dài khoảng thời gian triệt tiêu mặc định cho toàn bộ **Notice::Type** mà không cần phải tạo toàn bộ **Notice::policy** mục nhập và thiết lập `$suppress_for` trường.

Nâng cao thông báo

Một tập lệnh phải đưa ra thông báo cho bất kỳ trường hợp nào mà người dùng có thể muốn được thông báo hoặc thực hiện hành động. Ví dụ: bất cứ khi nào tập

lệnh phân tích SSH cơ sở thấy đủ lần đăng nhập không thành công vào một máy chủ nhất định, nó sẽ đưa ra thông báo về loại **SSH::Password_Guessing**. Mã trong tập lệnh phân tích SSH cơ sở đưa ra thông báo trông giống như sau:

```
NOTICE([$note=Password_Guessing,
        $msg=fmt("%s appears to be guessing SSH passwords (seen in %d connections).", key$host, r$num),
        $src=key$host,
      $identifier=cat(key$host)]);
```

NOTICE là một hàm bình thường trong không gian tên chung bao bọc một chức năng trong không gian tên Thông báo. Nó nhận một đối số duy nhất của **Notice::Info** kiểu bản ghi. Các trường phổ biến nhất được sử dụng khi đưa ra thông báo được mô tả trong bảng sau:

Tên trường	Sự mô tả
$note	Trường này là bắt buộc và là một giá trị enum đại diện cho kiểu thông báo.
$msg	Đây là thông báo con người có thể đọc được nhằm cung cấp thêm thông tin về trường hợp cụ thể của loại thông báo này.
$sub	Đây là một thông báo phụ dành cho con người có thể đọc được nhưng cũng sẽ thường xuyên được sử dụng để chứa dữ liệu phù hợp với **Notice::policy**.
$conn	Nếu bản ghi kết nối khả dụng khi thông báo được đưa ra và thông báo đại diện cho một số thuộc tính của kết nối, thì bản ghi kết nối có thể được đưa ra ở đây. Các trường khác như $ id và $ src sẽ tự động được điền từ giá trị này.
$id	Nếu một **conn_id** bản ghi có sẵn khi thông báo được đưa ra và thông báo đại diện cho một số thuộc tính của kết nối, thì kết nối có thể được đưa ra ở đây. Các trường khác như $src sẽ tự động được điền từ giá trị này.
$src	Nếu thông báo đại diện cho một thuộc tính của một máy chủ thì có thể chỉ trường này được điền để đại diện cho máy chủ đang được "chú ý".

Tên trường	Sự mô tả
$n	Điều này thường đại diện cho một số nếu thông báo liên quan đến một số. Nó thường được sử dụng nhất cho các bài kiểm tra số trong việc **Notice::policy** đưa ra các quyết định chính sách.
$identifier	Điều này đại diện cho một định danh duy nhất cho thông báo này. Trường này được mô tả chi tiết hơn trong phần Tự động triệt tiêu .
$suppress_for	Trường này có thể được đặt nếu có khoảng thời gian triệt tiêu tự nhiên cho thông báo có thể khác với giá trị mặc định. Giá trị được đặt cho trường này cũng có thể được người dùng sửa đổi **Notice::policy** vì vậy giá trị không được đặt vĩnh viễn và không thay đổi.

Khi viết các tập lệnh Zeek đưa ra thông báo, cần suy nghĩ một số điều về thông báo đại diện và dữ liệu nào nên được cung cấp để cung cấp cho người tiêu dùng thông báo thông tin tốt nhất về thông báo. Nếu thông báo đại diện cho nhiều kết nối và là một thuộc tính của máy chủ lưu trữ (ví dụ: máy chủ quét) thì có lẽ hợp lý nhất khi điền vào $src trường và không đưa ra kết nối hoặc **conn_id**. Nếu một thông báo đại diện cho một thuộc tính kết nối (ví dụ: đăng nhập SSH rõ ràng) thì bạn nên điền vào một trong hai $conn hoặc $id dựa trên dữ liệu có sẵn khi thông báo được đưa ra.

Sử dụng cẩn thận khi chèn dữ liệu vào thông báo sẽ giúp việc phân tích sau này dễ dàng hơn khi chỉ có dữ liệu để thể hiện đầy đủ sự kiện đã nêu ra thông báo. Ví dụ: nếu thông tin kết nối đầy đủ được bao gồm khi chứng chỉ máy chủ SSL sắp hết hạn, các nhật ký sẽ rất khó hiểu vì kết nối mà chứng chỉ được phát hiện là một chủ đề phụ với thực tế là chứng chỉ đã hết hạn đã được phát hiện. Trong nhiều trường hợp, có thể cần tạo hai hoặc nhiều thông báo riêng biệt. Ví dụ: một có thể là để phát hiện chứng chỉ SSL đã hết hạn và một có thể là để phát hiện nếu khách hàng quyết định tiếp tục với kết nối mà bỏ qua chứng chỉ đã hết hạn.

Tự động triệt tiêu

Khung thông báo hỗ trợ ngăn chặn các thông báo nếu tác giả của tập lệnh tạo ra thông báo đã chỉ ra cho khung thông báo cách xác định các thông báo về bản chất là giống nhau. Việc xác định các thông báo "về bản chất là trùng lặp" này được thực hiện với một trường tùy chọn trong **Notice::Info** các bản ghi có tên $identifier là một chuỗi đơn giản. Nếu các trường $identifier và $note trường giống nhau đối với hai thông báo, thì khung thông báo thực sự coi chúng là cùng một thứ và có thể sử dụng thông tin đó để loại bỏ các trường trùng lặp trong một khoảng thời gian có thể định cấu hình.

Trường $identifier này thường bao gồm một số phần dữ liệu liên quan đến thông báo mà khi kết hợp lại đại diện cho một phiên bản duy nhất của thông báo đó. Dưới đây là một ví dụ về script policy / protocols / ssl / validate-certs.zeek đưa ra thông báo cho các cuộc thương lượng phiên trong đó chứng chỉ hoặc chuỗi chứng chỉ không xác thực thành công so với các chứng chỉ của tổ chức phát hành chứng chỉ có sẵn.

```
NOTICE([$note=SSL::Invalid_Server_Cert,
        $msg=fmt("SSL certificate validation failed with (%s)", c$ssl$validation_status),
        $sub=c$ssl$subject,
        $conn=c,
        $identifier=cat(c$id$resp_h,c$id$resp_p,c$ssl$validation_status,c$ssl$cert_hash)]);
```

Trong ví dụ trên, bạn có thể thấy rằng $identifier trường chứa một chuỗi được tạo từ địa chỉ IP của người trả lời và cổng, thông báo trạng thái xác thực và tổng MD5 của chứng chỉ máy chủ. Các trường đó cụ thể được chọn vì các chứng chỉ SSL khác nhau có thể được nhìn thấy trên bất kỳ cổng nào của máy chủ, chứng chỉ có thể không xác thực được vì các lý do khác nhau và nhiều chứng chỉ máy chủ có thể được sử dụng trên sự kết hợp của địa chỉ IP và cổng với phần mở rộng SSL server_name (giải thích việc bổ sung tổng MD5 của chứng chỉ). Kết quả là nếu chứng chỉ không được xác thực và tất cả bốn phần dữ liệu đều khớp (địa chỉ IP, cổng, trạng thái xác thực và băm chứng chỉ) thì thông báo cụ thể đó sẽ không được đưa ra nữa trong khoảng thời gian chặn mặc định.

Việc đặt `$identifier` trường được để cho những người đưa ra thông báo vì người ta cho rằng tác giả tập lệnh đang đưa ra thông báo hiểu toàn bộ vấn đề và các trường hợp cạnh của thông báo mà người dùng có thể không thấy rõ. Nếu người dùng không muốn việc loại bỏ diễn ra hoặc đơn giản là muốn một khoảng thời gian khác, họ có thể đặt khoảng thời gian loại bỏ thông báo thành `0secs` hoặc xóa giá trị khỏi `$identifier` trường trong một dấu `Notice::policy` móc.

- **Mở rộng khung thông báo**

Có một số cơ chế để mở rộng khung thông báo và thêm khả năng mới.

Định cấu hình email thông báo

Nếu `Notice::mail_dest` được đặt, các thông báo với một hành động e-mail liên quan sẽ được gửi đến địa chỉ đó. Để có thêm tùy chỉnh, người dùng có thể sử dụng `Notice::policy` hook để sửa đổi trường *email_dest* . Ví dụ sau sẽ tạo ra 3 e-mail riêng biệt:

```
hook Notice::policy(n: Notice::Info)
  {
  n$email_dest = set(
      "snow.white@example.net",
      "doc@example.net",
      "happy@example.net,sleepy@example.net,bashful@example.net"
  );
  }
```

Nếu có thêm thông tin mà bạn muốn thêm vào email, bạn có thể thêm thông tin đó bằng cách viết `Notice::policy` hook.

Có một trường trong `Notice::Info` bản ghi có tên *$ email_body_section* sẽ được bao gồm nguyên văn khi email được gửi. Dưới đây là một ví dụ về việc bao gồm một số thông tin từ một yêu cầu HTTP.

```
hook Notice::policy(n: Notice::Info)
  {
  if ( n?$conn && n$conn?$http && n$conn$http?$host )
    n$email_body_sections[|n$email_body_sections|] = fmt("HTTP host header: %s", n$conn$http$host);
  }
```

Xem xét cụm

Khi chạy Zeek trong một cụm, hầu hết thông tin ở trên được giữ nguyên. Thông báo được tạo, **Notice::policy** hook được đánh giá và mọi hành động được chạy trên nút tạo ra thông báo (thường là nút worker). Lưu ý đối với người dùng / nhà phát triển của Zeek là mọi tệp hoặc quyền truy cập cần thiết để chạy các hành động thông báo phải có sẵn cho (các) nút tương ứng.

Vai trò của người quản lý là nhận và phân phối thông tin ngăn chặn thông báo, để các thông báo trùng lặp không được tạo ra. Hãy nhớ rằng có một số độ trễ nội tại trong quá trình đồng bộ hóa này, vì vậy có thể các thông báo tạo nhanh sẽ được lặp lại (và trong trường hợp này, bất kỳ hành động nào sẽ được thực hiện nhiều lần, một lần bởi mỗi nhân viên tạo thông báo).

Nhật ký kỳ lạ

Một loạt các hoạt động "kỳ lạ" được Zeek phát hiện có thể kích hoạt các sự kiện tương ứng thông báo cho lớp kịch bản về hoạt động này. Các sự kiện này tồn tại ở nhiều mức độ chi tiết khác nhau, bao gồm , và **conn_weird** các sự kiện khác. Được xây dựng trên đỉnh khung thông báo, mô-đun Weird triển khai các trình xử lý sự kiện có chức năng chuyển các "điểm bất thường" khác nhau vào các trình xử lý khung thông báo thông thường. Để có ý tưởng về các kiểu kỳ lạ có sẵn, hãy xem **flow_weirdnet_weirdfile_weirdWeird::actions** bảng, xác định các hành động mặc định cho các loại hoạt động khác nhau. Weirds thường không chỉ ra hoạt động liên quan đến bảo mật - chúng chỉ là những điều kỳ lạ mà bạn thường không mong đợi xảy ra, chẳng hạn như vi phạm máy trạng thái TCP kỳ lạ, chòm sao tiêu đề HTTP không mong muốn hoặc thuộc tính thông báo DNS nằm ngoài các thông số kỹ thuật RFC có liên quan. Đó là, đừng coi chúng là những phát hiện có thể hành

động theo nghĩa IDS, mặc dù chúng cũng có thể cung cấp manh mối bổ sung có ý nghĩa cho một sự cố bảo mật.

Loại thông báo cho các điểm bất thường là Activity. Bạn có nhiều hành động theo ý mình để biết cách xử lý các điểm bất thường: bạn có thể bỏ qua chúng, ghi lại chúng hoặc yêu cầu chúng kích hoạt thông báo, tất cả ở nhiều mức độ giảm / lọc khác nhau (xem các **Weird::Action** giá trị enum để biết thêm chi tiết). Đối với lọc động, bộ **Weird::ignore_hosts** và **Weird::weird_ignore** cho phép loại trừ hoạt động khỏi báo cáo.

❖ ***Framework Chữ ký***

Zeek chủ yếu dựa vào ngôn ngữ kịch bản mở rộng của nó để xác định và phân tích các chính sách phát hiện, nhưng nó cũng cung cấp một ngôn ngữ chữ ký độc lập để thực hiện đối sánh mẫu kiểu Snort cấp thấp. Mặc dù chữ ký không phải là công cụ phát hiện ưa thích của Zeek, nhưng đôi khi chúng có ích và gần với những gì mà nhiều người quen thuộc khi sử dụng các NIDS khác. Trang này cung cấp một cái nhìn tổng quan ngắn gọn về chữ ký của Zeek và bao gồm một số điểm tinh tế kỹ thuật của họ.

- **Khái niệm cơ bản**

Trước tiên, hãy xem xét một chữ ký ví dụ:

```
signature my-first-sig {
    ip-proto == tcp
    dst-port == 80
    payload /.*root/
    event "Found root!"
}
```

Chữ ký này yêu cầu Zeek so khớp biểu thức chính quy .*root trên tất cả các kết nối TCP đi đến cổng 80. Khi chữ ký kích hoạt, Zeek sẽ đưa ra một sự kiện **signature_match** có dạng:

```
event signature_match(state: signature_state, msg: string, data: string)
```

Ở đây, state chứa thêm thông tin về kết nối đã kích hoạt đối sánh, msg là chuỗi được chỉ định bởi câu lệnh sự kiện của chữ ký ( ) và dữ liệu là phần trọng tải cuối cùng đã kích hoạt đối sánh mẫu. Found root!

Vì chữ ký độc lập với các tập lệnh của Zeek, chúng được đưa vào (các) tệp của riêng họ. Có ba cách để chỉ định tệp nào chứa chữ ký: Bằng cách sử dụng -s cờ khi bạn gọi Zeek hoặc bằng cách mở rộng biến Zeek **signature_files** bằng += toán tử hoặc bằng cách sử dụng @load-sigs chỉ thị bên trong tập lệnh Zeek. Nếu một tệp chữ ký được cung cấp mà không có đường dẫn đầy đủ, nó sẽ được tìm kiếm theo cách bình thường ZEEKPATH. Ngoài ra, @load-sigs chỉ thị có thể được sử dụng để tải các tệp chữ ký trong một đường dẫn liên quan đến tập lệnh Zeek mà nó được đặt, ví dụ: sẽ mong đợi tệp chữ ký đó trong cùng thư mục với tập lệnh Zeek. Phần mở rộng mặc định của tên tệp là và Zeek sẽ tự động thêm phần mở rộng đó khi cần thiết. @load-sigs ./mysigs.sig.sig

- **Ngôn ngữ Chữ ký cho Lưu lượng Mạng**

Chúng ta hãy xem xét định dạng của một chữ ký kỹ hơn. Mỗi chữ ký cá nhân có định dạng, đâu là nhãn duy nhất cho chữ ký. Có hai loại thuộc tính: *điều kiện* và *hành động*. Các điều kiện xác định thời điểm chữ ký khớp, trong khi các hành động khai báo những việc cần làm trong trường hợp khớp. Các điều kiện có thể được chia thành bốn loại: *tiêu đề, nội dung , phụ thuộc* và *ngữ cảnh* . Chúng tôi thảo luận về tất cả những điều này chi tiết hơn trong phần sau. signature <id> { <attributes> }<id>

Các điều kiện

- Điều kiện tiêu đề

Điều kiện tiêu đề giới hạn khả năng áp dụng của chữ ký cho một tập con lưu lượng có chứa tiêu đề gói phù hợp. Loại đối sánh này chỉ được thực hiện cho gói đầu tiên của một kết nối.

Có các điều kiện tiêu đề được xác định trước cho một số trường tiêu đề được sử dụng nhiều nhất. Tất cả chúng thường có định dạng, nơi đặt tên trường tiêu đề và là danh sách các giá trị được phân tách bằng dấu phẩy hoặc phạm vi giá trị để so sánh (ví dụ: đối với các số từ 5 đến 10, không bao gồm 6). Các từ khóa sau được xác định: `<keyword> <cmp> <value-list><keyword>cmp==!=<=>=<value-list>5,7-10`

src-ip / **dst-ip <cmp> <address-list>**

Địa chỉ nguồn và địa chỉ đích, tương ứng. Địa chỉ có thể được cung cấp dưới dạng địa chỉ IPv4 hoặc IPv6 hoặc mặt nạ CIDR. Đối với địa chỉ IPv6 / mặt nạ, biểu thị dấu hai chấm-thập lục phân của địa chỉ phải được đặt trong dấu ngoặc vuông (ví dụ: `[fe80::1]` hoặc `[fe80::0]/16`).

src-port / **dst-port <cmp> <int-list>**

Cổng nguồn và cổng đích tương ứng.

ip-proto <cmp> tcp|udp|icmp|icmp6|ip|ip6

Trường Giao thức của tiêu đề IPv4 hoặc trường Tiêu đề tiếp theo của tiêu đề IPv6 cuối cùng (tức là trường Tiêu đề tiếp theo trong tiêu đề IPv6 cố định nếu không có tiêu đề tiện ích mở rộng hoặc trường đó từ tiêu đề tiện ích mở rộng cuối cùng trong chuỗi). Lưu ý rằng các dạng đường hầm IP-trong-IP được tự động giải mã theo mặc định và chữ ký chỉ áp dụng cho gói bên trong nhất, vì vậy chỉ định `ip` hoặc `ip6` là không.

Đối với danh sách nhiều giá trị, chúng được so sánh tuần tự với trường tiêu đề tương ứng. Nếu ít nhất một trong các phép so sánh là true, thì toàn bộ điều kiện tiêu đề sẽ khớp (ngoại lệ: với `!=`, điều kiện tiêu đề chỉ khớp nếu tất cả các giá trị khác nhau).

Ngoài các từ khóa tiêu đề được xác định trước này, điều kiện tiêu đề chung có thể được xác định là:

```
header <proto>[<offset>:<size>] [& <integer>] <cmp> <value-list>
```

Điều này so sánh giá trị được tìm thấy tại vị trí nhất định của tiêu đề gói với một danh sách các giá trị. Offset xác định vị trí của giá trị trong tiêu đề của giao thức được xác định bởi (có proto thể là ip, hoặc) . là 1, 2 hoặc 4 và chỉ định giá trị có kích thước bằng nhiều byte này. Nếu tùy chọn được đưa ra, giá trị của gói trước tiên được che bằng số nguyên trước khi nó được so sánh với danh sách giá trị là danh sách các số nguyên được phân tách bằng dấu phẩy hoặc phạm vi số nguyên tương tự như các số nguyên được mô tả ở trên. Các số nguyên trong danh sách có thể được theo sau bởi một phần bổ sung ở đó ip6tcpudpicmpicmp6size& <integer>cmp==!=<<=>>=value-list/ maskmask là một giá trị từ 0 đến 32. Giá trị này tương ứng với ký hiệu CIDR cho mặt nạ mạng và được dịch thành một mặt nạ bit tương ứng áp dụng cho giá trị của gói trước khi so sánh (tương tự như tùy chọn). Giá trị địa chỉ IPv6 không được phép trong danh sách giá trị, mặc dù bạn vẫn có thể kiểm tra bất kỳ phần 1, 2 hoặc 4 byte nào của tiêu đề IPv6 bằng từ khóa này. & integer

Đặt tất cả lại với nhau, đây là một điều kiện ví dụ tương đương với : dst-ip == 1.2.3.4/16, 5.6.7.8/24

```
header ip[16:4] == 1.2.3.4/16, 5.6.7.8/24
```

Lưu ý rằng ví dụ tương tự cho IPv6 hiện không khả thi vì 4 byte là chiều rộng tối đa của một giá trị có thể được so sánh.

- Điều kiện nội dung

Các điều kiện nội dung được xác định bởi các biểu thức chính quy. Chúng tôi phân biệt hai loại điều kiện nội dung: thứ nhất, biểu thức có thể được khai báo với payload câu lệnh, trong trường hợp đó, nó được so khớp với trọng tải thô của kết nối (đối với các luồng TCP được tập hợp lại) hoặc của từng gói (đối với ICMP, UDP và non - TCP được lắp ráp lại). Thứ hai, nó có thể được bắt đầu bằng một nhãn

dành riêng cho máy phân tích, trong trường hợp đó, biểu thức được khớp với dữ liệu được trích xuất bởi máy phân tích tương ứng.

Một `payload` điều kiện có dạng:

```
payload /<regular expression>/
```

Hiện tại, các điều kiện nội dung cụ thể cho máy phân tích sau được xác định (lưu ý rằng máy phân tích tương ứng phải được kích hoạt bằng cách tải tập lệnh chính sách của nó):

`http-request /<regular expression>/`

Biểu thức chính quy được so khớp với các URI được giải mã của các yêu cầu HTTP. Bí danh lỗi thời `http`:.

`http-request-header /<regular expression>/`

Biểu thức chính quy được so khớp với tiêu đề HTTP phía máy khách.

`http-request-body /<regular expression>/`

Biểu thức chính quy được so khớp với nội dung phía máy khách của các yêu cầu HTTP.

`http-reply-header /<regular expression>/`

Biểu thức chính quy được so khớp với tiêu đề HTTP phía máy chủ.

`http-reply-body /<regular expression>/`

Biểu thức chính quy được so khớp với nội dung phản hồi HTTP phía máy chủ.

`ftp /<regular expression>/`

Biểu thức chính quy được so khớp với đầu vào dòng lệnh của các phiên FTP.

`finger /<regular expression>/`

Biểu thức chính quy được so khớp với yêu cầu ngón tay.

Ví dụ: khớp với bất kỳ URI nào có chứa hoặc . Để lọc các loại yêu cầu, ví dụ: sử dụng . `http-request /.*(etc/(passwd|shadow)/etc/passwdetc/shadowGETpayload /GET /`

Lưu ý rằng kết nối HTTP (nghĩa là nhiều giao dịch HTTP trong một kết nối TCP duy nhất) có một số tác dụng phụ đối với kết quả khớp chữ ký. Nếu nhiều điều kiện được chỉ định trong một chữ ký, thì chữ ký này sẽ khớp nếu tất cả các điều kiện được đáp ứng bởi bất kỳ giao dịch HTTP nào (không nhất thiết phải luôn giống nhau!) Trong một kết nối pipelined.

- Điều kiện phụ thuộc

Để xác định sự phụ thuộc giữa các chữ ký, có hai điều kiện:

`requires-signature [!] <id>`

Xác định chữ ký hiện tại để khớp chỉ khi chữ ký được cung cấp `id` khớp với cùng một kết nối. Sử dụng `!` phủ định điều kiện: Chữ ký hiện tại chỉ khớp nếu `id` không khớp với cùng một kết nối (sử dụng điều này sẽ xác định quyết định đối sánh cho đến khi kết nối chấm dứt).

`requires-reverse-signature [!] <id>`

Tương tự với `requires-signature`, nhưng `id` phải khớp với chiều ngược lại của cùng một kết nối, so với chữ ký hiện tại. Điều này cho phép mô hình hóa khái niệm yêu cầu và trả lời.

- Điều kiện ngữ cảnh

Các điều kiện bối cảnh chuyển quyết định trận đấu cho các thành phần khác của Zeek. Chúng chỉ được đánh giá nếu tất cả các điều kiện khác đã phù hợp. Các điều kiện ngữ cảnh sau được xác định:

`eval <policy-function>`

Hàm chính sách đã cho được gọi và phải trả về một boolean xác nhận khớp. Nếu trả về false, sẽ không có kết quả khớp chữ ký nào được kích hoạt. Hàm phải thuộc loại. Ở đây, có thể chứa đoạn nội dung gần đây nhất có sẵn tại thời điểm chữ ký được khớp. Nếu không có đoạn nào như vậy, sẽ là chuỗi trống. Function cond(state: signature_state, data: string): booldatadata **signature_state**

payload-size <cmp> <integer>

So sánh số nguyên với kích thước của khối lượng gói tin. Đối với các luồng TCP được tập hợp lại, số nguyên được so sánh với kích thước của đoạn trọng tải theo thứ tự đầu tiên. Lưu ý rằng cái sau không được xác định rõ ràng.

same-ip

Đánh giá thành true nếu địa chỉ nguồn của các gói IP bằng với địa chỉ đích của nó.

tcp-state <state-list>

Áp đặt các hạn chế đối với trạng thái TCP hiện tại của kết nối. state-list là danh sách các từ khóa được phân tách bằng dấu phẩy established (bắt tay ba bước đã được thực hiện), originator (dữ liệu hiện tại được gửi bởi người khởi tạo kết nối) và responder (dữ liệu hiện tại được gửi bởi người phản hồi của kết nối).

udp-state <state-list>

Áp đặt các hạn chế về hướng luồng UDP phù hợp. state-list là danh sách được phân tách bằng dấu phẩy của một trong hai originator (dữ liệu hiện tại được gửi bởi người khởi tạo kết nối) hoặc responder (dữ liệu hiện tại được gửi bởi người phản hồi của kết nối). Trạng established thái bị từ chối là một lỗi trong chữ ký vì nó không có ý nghĩa hữu ích như đối với TCP.

- Hành động

Các thao tác xác định việc cần làm nếu một chữ ký khớp. Hiện tại, có hai hành động được xác định:

event <string>

Nâng cao một **signature_match** sự kiện. Trình xử lý sự kiện có loại sau:

event signature_match(state: signature_state, msg: string, data: string)
Chuỗi đã cho được chuyển vào dưới dạng msg và dữ liệu là phần hiện tại của trọng tải cuối cùng dẫn đến khớp chữ ký (điều này có thể trống đối với chữ ký không có điều kiện nội dung).

enable <string>

Bật trình phân tích giao thức <string> cho kết nối phù hợp ("http", "ftp" v.v.). Điều này được sử dụng bởi tính năng phát hiện giao thức động của Zeek để kích hoạt các bộ phân tích khi đang di chuyển.

- **Ngôn ngữ Chữ ký cho Nội dung Tệp**

Khung chữ ký cũng có thể được sử dụng để xác định các loại tệp MIME bất kể giao thức / kết nối mạng mà tệp được truyền qua đó. Một loại chữ ký đặc biệt có thể được viết cho mục đích này và sẽ được sử dụng tự động bởi Files Framework hoặc các tập lệnh Zeek sử dụng **file_magic** chức năng tích hợp sẵn.

- Các điều kiện

Chữ ký tệp sử dụng một loại điều kiện nội dung duy nhất ở dạng biểu thức chính quy:

file-magic /<regular expression>/

Điều này tương tự với payload điều kiện nội dung cho ngôn ngữ chữ ký lưu lượng mạng được mô tả ở trên. Sự khác biệt là payload chữ ký được áp dụng cho tải

trọng của kết nối mạng, nhưng file-magic có thể được áp dụng cho bất kỳ dữ liệu tùy ý nào, nó không cần phải ràng buộc với một giao thức / kết nối mạng.

- Hành động

Khi khớp một đoạn dữ liệu, chữ ký tệp sử dụng hành động sau để nhận thông tin về kiểu MIME của dữ liệu đó:

file-mime <string> [, <integer>]

Các đối số bao gồm chuỗi kiểu MIME được liên kết với biểu thức chính quy ma thuật tệp và "độ mạnh" tùy chọn dưới dạng số nguyên có dấu. Vì nhiều chữ ký ma thuật của tệp có thể khớp với một đoạn dữ liệu nhất định, nên giá trị sức mạnh có thể được sử dụng để giúp chọn "người chiến thắng". Giá trị cao hơn được coi là mạnh hơn.

❖ ***Framework đầu vào***

Zeek có khung nhập liệu linh hoạt cho phép người dùng nhập dữ liệu tùy ý vào Zeek. Dữ liệu được đọc vào các bảng Zeek hoặc được chuyển đổi trực tiếp thành các sự kiện để các tập lệnh xử lý khi chúng thấy phù hợp. Kiến trúc trình đọc mô-đun cho phép đọc từ tệp, cơ sở dữ liệu hoặc các nguồn dữ liệu khác.

Đọc dữ liệu thành bảng

Tệp đầu vào được đọc vào bảng với lệnh gọi hàm: **Input::add_table**

```
global denylist: table[addr] of Val = table();

event zeek_init() {
    Input::add_table([$source="denylist.file", $name="denylist",
                      $idx=Idx, $val=Val, $destination=denylist]);
    Input::remove("denylist");
}
```

Với ba dòng này, trước tiên, chúng ta tạo một bảng trống sẽ nhận dữ liệu danh sách từ chối và sau đó hướng dẫn khung công tác đầu vào mở một luồng đầu vào có tên "danh sách từ chối" để đọc dữ liệu vào bảng. Dòng thứ ba lại loại bỏ luồng đầu vào, vì chúng ta không cần nó nữa sau khi dữ liệu đã được đọc.

- Xử lý không đồng bộ

Vì một số tệp dữ liệu có thể khá lớn, khung đầu vào hoạt động không đồng bộ. Một luồng mới được tạo cho mỗi luồng đầu vào mới. Luồng này mở tệp dữ liệu đầu vào, chuyển đổi dữ liệu thành định dạng bên trong và gửi nó trở lại luồng Zeek chính. Do đó, dữ liệu không thể truy cập ngay lập tức. Tùy thuộc vào kích thước của nguồn dữ liệu, có thể mất từ vài mili giây đến vài giây cho đến khi tất cả dữ liệu có trong bảng.

Các lệnh gọi tiếp theo đến một nguồn đầu vào được xếp hàng đợi cho đến khi hoàn thành hành động trước đó. Vì điều này, ví dụ, nó có thể được gọi **Input::add_table** và **Input::remove** trong hai dòng tiếp theo: hành động loại bỏ sẽ vẫn được xếp hàng đợi cho đến khi hoàn thành lần đọc đầu tiên.

Khi khung công tác đầu vào đọc xong từ một nguồn dữ liệu, nó sẽ kích hoạt **Input::end_of_data** sự kiện. Khi sự kiện này đã được nhận, tất cả dữ liệu từ tệp đầu vào sẽ có sẵn trong bảng.

```
event Input::end_of_data(name: string, source: string) {
    # now all data is in the table
    print denylist;
}
```

Bảng có thể được sử dụng trong khi dữ liệu vẫn đang được đọc - nó chỉ có thể không chứa tất cả các dòng từ tệp đầu vào trước khi sự kiện kích hoạt. Sau khi bảng đã được điền, nó có thể được sử dụng giống như bất kỳ bảng Zeek nào khác và các mục từ chối danh sách có thể dễ dàng được kiểm tra:

```
if ( 192.168.18.12 in denylist )
    # take action
```

- Bộ thay vì bảng

Đối với một số trường hợp sử dụng, khái niệm khóa / giá trị thúc đẩy dữ liệu dạng bảng không áp dụng, chẳng hạn như khi mục đích chính của dữ liệu là để kiểm tra tư cách thành viên trong một tập hợp. Khung đầu vào hỗ trợ phương pháp này bằng cách sử dụng các tập hợp làm kiểu dữ liệu đích và bỏ `$val` qua **`Input::add_table`**:

```
type Idx: record {
    ip: addr;
};

global denylist: set[addr] = set();

event zeek_init() {
    Input::add_table([$source="denylist.file", $name="denylist",
                      $idx=Idx, $destination=denylist]);
    Input::remove("denylist");
}
```

- Đọc lại và phát trực tuyến dữ liệu

Đối với một số nguồn dữ liệu (chẳng hạn như nhiều danh sách từ chối), dữ liệu đầu vào thay đổi liên tục. Khung đầu vào hỗ trợ các kỹ thuật bổ sung để quản lý đầu vào luôn thay đổi như vậy.

Phương pháp đầu tiên, rất cơ bản là làm mới một cách rõ ràng một luồng đầu vào. Khi một luồng đầu vào đang mở (nghĩa là nó vẫn chưa bị loại bỏ bởi một lệnh gọi đến **`Input::remove`**), hàm **`Input::force_update`** có thể được gọi. Điều này sẽ kích hoạt làm mới hoàn toàn bảng: mọi phần tử đã thay đổi từ tệp sẽ được cập nhật, những phần tử mới được thêm vào và bất kỳ phần tử nào không còn trong dữ liệu đầu vào sẽ bị xóa. Sau khi cập nhật xong, **`Input::end_of_data`** sự kiện sẽ được nâng lên.

- Nhận các sự kiện thay đổi

Khi đọc lại tệp, có thể thú vị khi biết chính xác dòng nào trong tệp nguồn đã thay đổi. Vì lý do này, khuôn khổ đầu vào có thể đưa ra một sự kiện mỗi khi một mục dữ liệu được thêm vào, xóa khỏi hoặc thay đổi trong bảng.

- Lọc dữ liệu trong quá trình nhập

Khung nhập liệu cũng cho phép người dùng lọc dữ liệu trong quá trình nhập. Để đạt được mục đích này, các chức năng vị ngữ được sử dụng. Một hàm vị từ được gọi trước khi một phần tử mới được thêm / thay đổi / xóa khỏi bảng. Vị từ có thể chấp nhận hoặc phủ quyết thay đổi bằng cách trả về true cho một thay đổi được chấp nhận và false cho một thay đổi bị từ chối. Hơn nữa, nó có thể thay đổi dữ liệu trước khi nó được ghi vào bảng.

- Dữ liệu đầu vào bị hỏng

Khung đầu vào thông báo cho bạn về các sự cố trong quá trình nhập dữ liệu theo hai cách. Đầu tiên, các thông báo của phóng viên, kết thúc trong report.log, cho biết loại sự cố và tệp mà sự cố đã xảy ra:

```
#fields ts      level   message location
0.000000        Reporter::WARNING       denylist.file/Input::READER_ASCII: Did not find requested field
```

Thứ hai, **Input::TableDescription** và **Input::EventDescription** các bản ghi có một $error_ev thành viên để kích hoạt các sự kiện chỉ ra cùng một thông báo và mức độ nghiêm trọng như được hiển thị ở trên. Việc sử dụng các sự kiện này phản ánh sự kiện thay đổi.

Đọc dữ liệu cho sự kiện

Chế độ nhập dữ liệu thứ hai của khung nhập liệu trực tiếp tạo ra các sự kiện Zeek từ dữ liệu đã nhập thay vì chèn chúng vào bảng. Luồng sự kiện hoạt động rất giống với các luồng bảng được thảo luận ở trên và hầu hết các tính năng được thảo luận (chẳng hạn như các vị từ để lọc) cũng hoạt động cho các luồng sự kiện.

Luồng sự kiện khác với luồng bảng theo hai cách:

- Luồng sự kiện không cần khai báo chỉ mục và giá trị riêng biệt - thay vào đó, tất cả các kiểu dữ liệu nguồn được cung cấp trong một định nghĩa bản ghi duy nhất.
- Vì khung công tác nhận thức một dòng sự kiện liên tục, nên nó không có khái niệm về đường cơ sở dữ liệu (ví dụ: một bảng) để so sánh dữ liệu đến. Do đó, loại sự kiện thay đổi (một **Input::Event** trường hợp, tpe ở trên) hiện luôn luôn **Input::EVENT_NEW**.

Người đọc dữ liệu

Khung đầu vào hỗ trợ các loại trình đọc khác nhau cho các loại tệp dữ liệu nguồn khác nhau. Hiện tại, khuôn khổ mặc định nhập các tệp ASCII được định dạng ở định dạng tệp nhật ký Zeek (các giá trị được phân tách bằng tab bằng #fields dòng tiêu đề). Một số trình đọc khác được bao gồm trong Zeek và các gói / plugin của Zeek có thể cung cấp các trình đọc bổ sung.

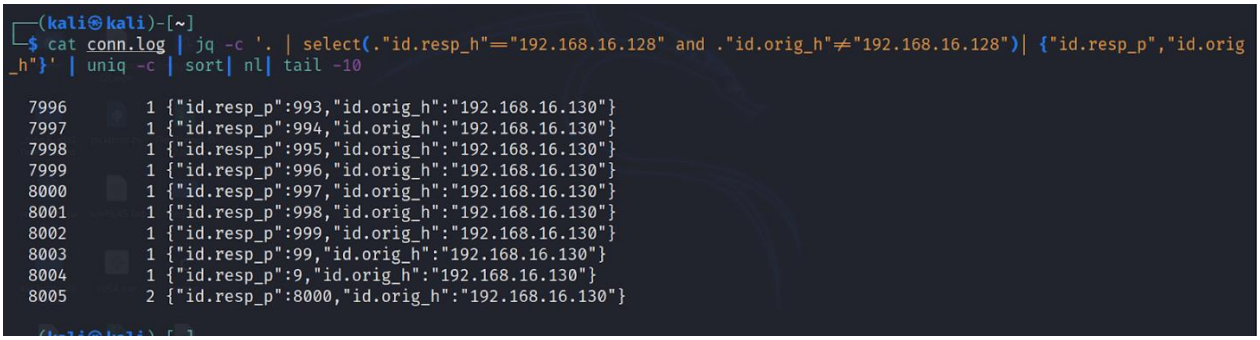
III. Demo

1. Kịch bản

Kẻ tấn công quét máy mục tiêu phát hiện dịch vụ web trên cổng 8000. Sau đó truy cập trang web và phát hiện lỗ hổng cho phép tải lên php. Kẻ tấn công tải lên file php và truy cập tạo reverse shell. Sau đó tải lên mã độc vào máy nạn nhân.

Kịch bản phân tích:

1) Nhận được cảnh báo bất thường khi có địa chỉ IP 192.168.16.130 thực hiện quét cổng nên ta sẽ phân tích log dựa trên hành vi này:
cat conn.log | jq -c '. | select(."id.resp_h"=="192.168.16.128" and ."id.orig_h"!="192.168.16.128")| {"id.resp_p","id.orig_h"}' | uniq -c | sort| nl | tail -10



2) Liệt kê IP thực hiện kết nối tcp
cat conn.log | jq -c '. | select(.proto=="tcp" and ."id.orig_h" != "192.168.16.128")| ."id.orig_h" '|uniq -c

3) Liệt kê kết nối tới cổng 8000
cat conn.log | jq -c '. | select(.proto=="tcp" and ."id.resp_p"==8000)| {ts,"id.orig_h",duration} '| uniq -c | sort

4) Kết nối http

cat http.log | jp -c'.|select(."id.resp_p"==8000 and ."id.resp_h" ===" 192.168.16.128")|{ts,"id.orig_h",method,uri}'

```
┌──(kali㉿kali)-[~]
└─$ cat http.log | jq -c '. | select(."id.resp_p"==8000 and ."id.resp_h"=="192.168.16.128")| {ts,"id.orig_h",method,uri}'
{"ts":1654431586.596322,"id.orig_h":"192.168.16.130","method":"GET","uri":"/"}
{"ts":1654431598.704117,"id.orig_h":"192.168.16.130","method":"POST","uri":"/upload.php"}
{"ts":1654431633.840398,"id.orig_h":"192.168.16.130","method":"GET","uri":"/uploads/reversess.php"}
```

5) Phát hiện file Malware

cat files.log| jq -c '. | select (.tx_hosts==["192.168.16.130"] and .source=="HTTP")| {mime_type,seen_bytes}'| uniq -c

2. IOC

- Scan cổng phát hiện quét theo số lượng cổng (15 cổng trở lên)
- Tải lên file PhP trong thư mục lưu trữ hình ảnh
- Truy cập file php ngay trong chính thư mục lưu trữ dữ liệu
- Kết nối ra bên ngoài với địa chỉ nằm trong blacklist
- Nhận biết malware dựa trên hash

```
┌──(kali㉿kali)-[~]
└─$ sudo zeek -C -i eth0 library_zeek.zeek
listening on eth0

{
[192.168.27.2] = [reason=Botnet server],
[192.168.16.130] = [reason=Malware host],
[192.168.250.3] = [reason=Virus detected]
}
{
[43asds1321dadba123adkasbd123213asbdaksdb] = [typehash=Trojan],
[c8cb4e823da6cc04ffdbdec0301515469d082bc4] = [typehash=Malware],
[42f9e10495d0447f7e61624f69c42a26cadfccbf] = [typehash=Malware],
[42f9e10495d0447f7e61624f69c42aasdbasdsad] = [typehash=RAT]
}
```

```
192.168.16.130 scanned at least 15 unique ports of host 192.168.16.128 in 0m0s
Blacklist IP detection connected which is 192.168.16.130 at port 8000/tcp
Blacklist IP detection connected which is 192.168.16.130 at port 8000/tcp
Blacklist IP detection connected which is 192.168.16.130 at port 8000/tcp
Suspicious upload file PHP with text/x-php
Suspicious upload file PHP with text/x-php
Blacklist IP detection connected which is 192.168.16.130 at port 8000/tcp
Access PHP in directory /var/www/images/uploads is critical (image only) from 192.168.16.130
Suspicious upload file PHP with application/zip
Malware detection with the hash: c8cb4e823da6cc04ffdbdec0301515469d082bc4 type: Malware \x0a More information: https://www.virustotal.c
om/en/search/?query=c8cb4e823da6cc04ffdbdec0301515469d082bc4
Suspicious upload file PHP with application/zip
```

```
# wget 192.168.16.130:8000/malwaree.rar
--2022-06-04 11:44:25--  http://192.168.16.130:8000/malwaree.rar
Connecting to 192.168.16.130:8000... connected.
HTTP request sent, awaiting response... 200 OK
Length: 210552 (206K) [application/vnd.rar]
Saving to: ‘malwaree.rar.4’

 0K .......... .......... .......... .......... .......... 24% 12.6M 0s
50K .......... .......... .......... .......... .......... 48% 22.0M 0s
100K .......... .......... .......... .......... .......... 72% 15.4M 0s
150K .......... .......... .......... .......... .......... 97% 19.3M 0s
200K .....                                                  100% 17.0M=0.01s

2022-06-04 11:44:25 (16.5 MB/s) - ‘malwaree.rar.4’ saved [210552/210552]
```

IV. Script cho kịch bản

1. accessPhP.zeek

```
@load base/protocols/conn
@load base/protocols/http
export{
    redef enum Notice::Type+={
        Match1
    };
    option match_file_php= /^+[a-zA-Z0-9]\.php*[0-9]/;
    option method="http";
    option host="example.com";
    option p="8000";
}
event http_request(c: connection, method: string, original_URI: string,
                                    unescaped_URI: string, version: string)
{
    local x: PatternMatchResult= match_pattern(c$http$uri, /^\/uploads\/[a-zA-Z0-9]+\.php[0-9]{0,1}/);
    #^\/uploads\/\w+.php\d*
    if(x$matched){
        NOTICE([$msg="PHP access in url is dangerous!",$note=Match1,$conn=c]);
        print fmt("Access PHP in directory /var/www/images/uploads is critical (image only) from %s"
                                    ,c$id$orig_h);
    }
}
```

2. blacklist.zeek

```
@load base/protocols/conn
@load base/frameworks/notice
@load base/frameworks/netcontrol/main.zeek
@load base/frameworks/notice/main
@load base/frameworks/netcontrol
@load policy/frameworks/netcontrol/catch-and-release
export {
redef enum Notice::Type += {
    Blacklist_IP,
    };
}
type Idx1: record {
    ip: addr;
};
type Val1: record {
    reason: string;
};
global denylist: table[addr] of Val1 = table();
event zeek_init() {
    Input::add_table([$source="data.file", $name="denylist",
                      $idx=Idx1, $val=Val1, $destination=denylist]);
    Input::remove("denylist");
}
event Input::end_of_data(name: string, source: string) {
    print denylist;
}
event connection_established(c: connection){
   if(c$id$orig_h in denylist){
    #create_log();
NOTICE([$note=Blacklist_IP,
$msg=fmt("Blacklist IP: %s detection has connected from your server!!!",c$id$orig_h)]);
print fmt("Blacklist IP detection connected which is %s at port %s",c$id$orig_h,c$id$resp_p);
NetControl::drop_address(192.168.16.130, 1 sec);
}
   }
```

3. malware_dev.zeek

```
@load base/frameworks/files
@load base/frameworks/notice
@load frameworks/files/hash-all-files
export {
    redef enum Notice::Type += {
        Match3
    };
    option match_file_types_1 = /application\/raj/ | /application\/zip/
                               |/application\/php/ | /application\/rar/;
}
type Idx: record {
    hash: string;
};
type Val: record {
    typehash: string;
};
global malist: table[string] of Val = table();
event zeek_init() {
    Input::add_table([$source="malist.file", $name="malist",
                      $idx=Idx, $val=Val, $destination=malist]);
    Input::remove("malist");
}
event Input::end_of_data(name: string, source: string) {
    # now all data is in the table
    print malist;
}
event file_hash(f: fa_file, kind: string, hash: string)
    {
    local  link = "https://www.virustotal.com/en/search/?query=%s";
    local virustotal= fmt(link,hash);
    if ( kind == "sha1" && match_file_types_1 in f$info$mime_type && f?$info && f$info?$mime_type && hash in malist)
        print fmt("Malware detection with the hash: %s type: Malware \n More information: %s ",hash, virustotal);
    }
```

4. portscan.zeek

```
@load base/frameworks/notice
@load base/frameworks/sumstats
@load base/utils/time
################################
module Scan;
################################
export {
    redef enum Notice::Type += {
        Address_Scan,

        Port_Scan,
    };
    const addr_scan_interval = 5min &redef;
    const port_scan_interval = 5min &redef;
    const addr_scan_threshold = 25.0 &redef;
    const port_scan_threshold = 15.0 &redef;
    global Scan::addr_scan_policy: hook(scanner: addr, victim: addr, scanned_port: port);
    global Scan::port_scan_policy: hook(scanner: addr, victim: addr, scanned_port: port);
}
event zeek_init() &priority=5
    {
    local r1: SumStats::Reducer = [$stream="scan.addr.fail", $apply=set(SumStats::UNIQUE)
                                   , $unique_max=double_to_count(addr_scan_threshold+2)];
    SumStats::create([$name="addr-scan",
                      $epoch=addr_scan_interval,
                      $reducers=set(r1),
                      $threshold_val(key: SumStats::Key, result: SumStats::Result) =
                        {
                        return result["scan.addr.fail"]$unique+0.0;
                        },
                      #$threshold_func=check_addr_scan_threshold,
                      $threshold=addr_scan_threshold,
```

```
                    #$threshold_func=check_addr_scan_threshold,
                    $threshold=addr_scan_threshold,
                    $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
                      {
                      local r = result["scan.addr.fail"];
                      local side = Site::is_local_addr(key$host) ? "local" : "remote";
                      local dur = duration_to_mins_secs(r$end-r$begin);
                      local message=fmt("%s scanned at least %d unique hosts on port %s in %s"
                      , key$host, r$unique, key$str, dur);
            print (message);
                      NOTICE([$note=Address_Scan,
                              $src=key$host,
                              $p=to_port(key$str),
                              $sub=side,
                              $msg=message,
                              $identifier=cat(key$host)]);
                      }]);

    # Note: port scans are tracked similar to: table[src_ip, dst_ip] of set(port);
    local r2: SumStats::Reducer = [$stream="scan.port.fail", $apply=set(SumStats::UNIQUE)
                                   , $unique_max=double_to_count(port_scan_threshold+2)];
    SumStats::create([$name="port-scan",
                    $epoch=port_scan_interval,
                    $reducers=set(r2),
                    $threshold_val(key: SumStats::Key, result: SumStats::Result) =
                      {
                      return result["scan.port.fail"]$unique+0.0;
                      },
                    $threshold=port_scan_threshold,
                    $threshold_crossed(key: SumStats::Key, result: SumStats::Result) =
                      {
                      local r = result["scan.port.fail"];
                      local side = Site::is_local_addr(key$host) ? "local" : "remote";
                      local dur = duration_to_mins_secs(r$end-r$begin);
                      local message = fmt("%s scanned at least %d unique ports of host %s in %s"
                      , key$host, r$unique, key$str, dur);
                      print (message);
                      NOTICE([$note=Port_Scan,
                              $src=key$host,
                              $dst=to_addr(key$str),
                              $sub=side,
                              $msg=message,
                              $identifier=cat(key$host)]);
                      }]);
    }

function add_sumstats(id: conn_id, reverse: bool)
    {
    local scanner      = id$orig_h;
    local victim       = id$resp_h;
    local scanned_port = id$resp_p;

    if ( reverse )
        {
        scanner      = id$resp_h;
        victim       = id$orig_h;
        scanned_port = id$orig_p;
        }

    if ( hook Scan::addr_scan_policy(scanner, victim, scanned_port) )
        SumStats::observe("scan.addr.fail", [$host=scanner, $str=cat(scanned_port)], [$str=cat(victim)]);

    if ( hook Scan::port_scan_policy(scanner, victim, scanned_port) )
        SumStats::observe("scan.port.fail", [$host=scanner, $str=cat(victim)], [$str=cat(scanned_port)]);
    }
```

```
function is_failed_conn(c: connection): bool
    {
    # Sr || ( (hR || ShR) && (data not sent in any direction) )
    if ( (c$orig$state == TCP_SYN_SENT && c$resp$state == TCP_RESET) ||
         (((c$orig$state == TCP_RESET && c$resp$state == TCP_SYN_ACK_SENT) ||
           (c$orig$state == TCP_RESET && c$resp$state == TCP_ESTABLISHED && "S" in c$history )
          ) && /[Dd]/ !in c$history )
       )
        return T;
    return F;
    }

function is_reverse_failed_conn(c: connection): bool
    {
    # reverse scan i.e. conn dest is the scanner
    # sR || ( (Hr || sHr) && (data not sent in any direction) )
    if ( (c$resp$state == TCP_SYN_SENT && c$orig$state == TCP_RESET) ||
         (((c$resp$state == TCP_RESET && c$orig$state == TCP_SYN_ACK_SENT) ||
           (c$resp$state == TCP_RESET && c$orig$state == TCP_ESTABLISHED && "s" in c$history )
          ) && /[Dd]/ !in c$history )
       )
        return T;
    return F;
    }

event connection_attempt(c: connection)
    {
    local is_reverse_scan = F;
    if ( "H" in c$history )
        is_reverse_scan = T;

    add_sumstats(c$id, is_reverse_scan);
    }
```

```
    }

event connection_attempt(c: connection)
    {
    local is_reverse_scan = F;
    if ( "H" in c$history )
        is_reverse_scan = T;

    add_sumstats(c$id, is_reverse_scan);
    }

event connection_rejected(c: connection)
    {
    local is_reverse_scan = F;
    if ( "s" in c$history )
        is_reverse_scan = T;

    add_sumstats(c$id, is_reverse_scan);
    }

event connection_reset(c: connection)
    {
    if ( is_failed_conn(c) )
        add_sumstats(c$id, F);
    else if ( is_reverse_failed_conn(c) )
        add_sumstats(c$id, T);
    }

event connection_pending(c: connection)
    {
    if ( is_failed_conn(c) )
        add_sumstats(c$id, F);
    else if ( is_reverse_failed_conn(c) )
        add_sumstats(c$id, T);
    }
```

5. sus_upload.zeek

```
@load base/frameworks/files
@load base/frameworks/notice
@load frameworks/files/hash-all-files

export {
    redef enum Notice::Type += {
        Match2
    };

    ## File types to attempt matching against the Malware Hash Registry.
    option match_file_types = /application\/raj/ |
                              /application\/zip/ |
                              /application\/php/ |
                              /application\/rar/ |
                          /text\/x-php/ ;
}
event file_hash(f: fa_file, kind: string, hash: string)
    {
    if (match_file_types in f$info$mime_type && f?$info && f$info?$mime_type){
        print fmt("Suspicious upload file PHP with %s",f$info$mime_type);
    }
}
```

Kết luận

Báo cáo trình bày tổng quan về Zeek và Zeek Scripts. Đối với Zeek Scripts nhóm chúng em chú trọng về

- Tập lệnh
- Trình xử lý sự kiện
- Cấu trúc dữ liệu
- Hook
- Framework

Đồng thời nhóm em cũng có thực hiện demo với hình ảnh kèm theo.

Vì thời gian nghiên cứu có hạn, trình độ hiểu biết của chúng em còn nhiều hạn chế nên bài báo cáo không tránh khỏi những thiếu sót, em rất mong nhận được sự góp ý quý báu của cô để báo cáo của nhóm em được hoàn thiện hơn.

Tài liệu tham khảo

[1] Github, Book of Zeek.

[2] T. U. o. T. a. S. Antonio, "ZEEK INSTRUSION DETECTION SERIES," Lab8, Texas, 03-13-2020.