Người thực hiện: Tạ Đình Sung

# I>BRO IDS

## 1.Một vài thông tin về bro

* Bro là một platform mã nguồn mở dùng để giám sát, phân tích lưu lượng mạng một cách thụ động. Nó kiểm tra tất cả lưu lượng vào một link để tìm các dấu hiệu đáng ngờ.
* Tuy nhiên, Bro hỗ trợ một loạt các nhiệm vụ phân tích lưu lượng ngay cả bên ngoài các domai, bao gồm cả các phép đo hiệu suất và giúp đỡ trouble-shooting.
* Tại sao lại dùng Bro ?
  + Phân tích mạnh mẽ, thiết kế thanh lịch
  + Lắng nghe thụ động, ứng phó chủ động
  + Khả năng mở rộng tới mạng 100G và lớn hơn
  + Dữ liệu có cấu trúc-log files
  + Cộng đồng phát triển sôi động
  + Phát triển lâu đời và được hỗ trợ
* Đặc điểm:
  + Sử dụng các event callback để xử lý
  + Dùng code,không sử dụng rules như snort nên việc xử lý data linh hoạt hơn
  + Hỗ trợ xử lý realtime trên các nền ngôn ngữ khác như python,ruby
  + Các opensource signals,plugins ít,được cộng đồng sử dụng không nhiều bằng Snort.

## 2.Đánh giá Bro:

* + Bro là 1 framework rất tốt cho việc phát triển IDS,nhưng nếu cài ở mặc định thì việc  
    phát hiện tấn công là ở mức rất cơ bản ( ddos,sqli,scanning…)
  + Bro là 1 framework rất tốt cho việc phát triển IDS,nhưng nếu cài ở mặc định thì việc  
    phát hiện tấn công là ở mức rất cơ bản ( ddos,sqli,scanning…)
  + Sử dụng nó phải viết code nhiều vì opensource của cộng đồng ít

## 3.Shellcode với Bro

* Nhận diện bằng signatures có sẵn.
* Mặc định chỉ có 2 signatures cho windows.
* Có thể sử dụng database của snort
* Không thể nhận diện shellcode đã encode
* Cần 1 công cụ hỗ trợ nhận diện shellcode chính xác hơn

# II>Unicorn

* Unicorn là một emulator framework đa nền tảng, đa kiến trúc,
* Các tính năng nổi bậc:
  + Đa kiến trúc: Arm, Arm64, M68K, Mips, Sparc, & X86 (include X86\_64).
  + API nhẹ, trực quan, đơn giản
  + Thực hiện bằng ngôn ngữ thuần C, với các binding cho Rust, Haskell, Ruby, Python, Java, Go, .NET, Delphi/Pascal & MSVC available.
  + Hỗ trợ trực tiếp cho Window và \*nix (với MAC OS, Linux, \*BSD, Solaris).
  + Hiệu suất cao
* Thích hợp cho 1 IDS để phân tích Shellcode của nhiều thiết bị.

# III>Kết hợp Bro và Unicorn

## 1.Nhận xét về 2 bộ công cụ

* Bro có khả năng monitor traffic,log,filter….
* Unicorn có khả năng emulator shellcode
* Cả hai đều hỗ trợ chạy realtime trên python
* Có thể nhận diện shellcode bằng sự kết hợp Bro và Unicorn

## 2.Đưa ra mô hình kết hợp:

BRO

Data,shellcode

alert

**shellcode**

**Unicorn**

**shellcode**

Cant detect

## 2.Mô tả hoạt động của mô hình

Khi dữ liệu đi vào mạng thì Bro sẽ bắt lại và đem bộ signatures của các shellcode để so sánh,nếu có signature trùng khớp thì gọi event alert\_shellcode để cảnh báo. Nếu như không trùng khớp với signature shellcode mà trùng khớp với signature nghi ngờ có shellcode thì bro sẽ gửi data đó cho Unicorn kiểm tra. Lúc này Unicorn sẽ lấy payload đó và emulator lại để kiểm tra xem data đó có phải là encoded shellcode hay không. Nếu đúng thì Unicorn sẽ gửi lại cho bro 1 event là alert\_shellcode. Nếu không phải thì không gửi gì.

## 3.Phân tích code

Phần code cho bro: (file: detector.bro)

*redef Communication::listen\_port = 7331/tcp;*

*redef Communication::nodes += {*

*["mipu"] = [$host = 127.0.0.1, $events = /alert\_shellcode/, $connect=F, $ssl=F]};*

phần code ở đây là để mở port(7331) và đăng kí các event alert\_shellcode giúp bro chuyển các payload cho unicorn kiểm tra payload phải là shellcode hay không.

*export {*

*const match\_shellcode=/(909090909090909090)|(5c62696e5c7368)|(909090E8C0)|(cd80)|(5831d20f05)|(0f05)/;*

*const nonascii = /[^\x00-\x7F]/;*

*}*

Khai báo các signal shellcode và nonascii

*global is\_shellcode: event(pay: string, do: int);*

Khai báo event is\_shellcode là global giúp cho checkshellcode.py có thể nhận được event này

*event alert\_shellcode(sc: string,syscall: string,arg1: string,arg2: string)*

Khai báo event alert\_shellcode để cho checkshell.py có thể call event này lên khi nó phát hiện shellcode

*event tcp\_packet(c: connection, is\_orig: bool, flags: string, seq: count, ack: count, len: count, payload: string)*

*{*

*if(payload !=""){*

*if(noneasii in payload)*

*{*

*payload=bytestring\_to\_hexstr(payload);*

*print fmt("detected nonascii in payload!!!\n hexpayload : %s \n", payload);*

*if(match\_shellcode in payload){*

*print "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$";*

*print "$$$$ BRO DETECTED SHELLCODE $$$$$";*

*print "$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$";*

*}*

*else*

*{*

*event is\_shellcode(payload,1);*

*}*

*}*

*else{*

*print fmt("payload: %s",payload);*

*}*

*}*

*}*

Event này để bắt data lên và phân tích,ở đây detector kiểm tra payload có là ascii,nếu đúng thì in ra payload và không cảnh báo,nếu không phải ascii thì tiếp tục so sánh với signals match\_shellcode nếu đúng thì cảnh báo shellcode nếu sai thì gửi qua cho event is\_shellcode dến checkshell.py để kiểm tra có phải là shellcode hay không.

*event connection\_SYN\_packet(c:connection,pkt:SYN\_packet)*

*{*

*print fmt("---------------------------");*

*print fmt("new connection from : %s:%s --> %s:%s",c$id$orig\_h,c$id$orig\_p,c$id$resp\_h,c$id$resp\_p);*

*print fmt("---------------------------");*

*}*

Event này để in ra nếu có kết nối được khởi tạo.

Phần code cho checkshell.py

*from unicorn import \**

*from unicorn.x86\_const import \**

import các thư viện cần thiết để sử dụng unicorn

*bc = Connection("127.0.0.1:7331")*

*print("Unicorn connected to Bro!!!\nGO-->\nGO--> ")*

*while True:*

*bc.processInput();*

khởi tạo kết nối đến Bro tại port 7331,và vòng lặp while true để luôn luôn lắng nghe

*@event*

*def is\_shellcode(input,do):*

*global payload*

*print("Input: %s",input)*

*payload = input.decode("hex")*

*check\_shell(payload)*

*print("sent")*

khai báo event is\_shellcode để checkshell.py xử lý event này khi bro gửi.

*def hook\_syscall(mu, user\_data):*

*global payload*

*rax = mu.reg\_read(UC\_X86\_REG\_RAX)*

*rdi = mu.reg\_read(UC\_X86\_REG\_RDI)*

*rip = mu.reg\_read(UC\_X86\_REG\_RIP)*

*if rax == 0x3b: # sys\_execv*

*try:*

*buf = mu.mem\_read(rdi, 8)*

*s=""*

*for i in buf:*

*s+=chr(i)*

*print(s)*

*k = ">>> 0x%x: interrupt 0x%x, SYS\_EXECVE .argument =%s"%(rip, rax,s)*

*print (k)*

*bc.send("alert\_shellcode",string(payload),string("SYS\_EXECVE"),string(s),string(""))*

*except UcError as e:*

*k = ">>> 0x%x: interrupt 0x%x .argument =unknow"%(rip, rax)*

*return k*

*else:*

*k = ">>> 0x%x: interrupt 0x%x"%(rip, rax)*

*print(k)*

*return k*

*mu.emu\_stop()*

Function này hook khi có syscall được gọi,ở đây chúng ta sẽ lấy giá trị của rax để biết đoạn shellcode đang gọi tới syscall nào, và rdi là tham số truyền vào cho syscall đó. Ở đây lấy 1 trường hợp đơn giản là rax=0x3b là syscall execv rất hay được sử dụng trong các shellcode.

Sau khi lấy được các thông tin này thì chương trình sẽ gửi ngược lại event “alert\_shellcode” cho bro để cảnh báo payload là shellcode.

# IV.Tiến hành kiểm tra và kết quả:

Lần 1: gửi dữ liệu bình thường không có shellcode:  
🡺 không có cảnh báo gì

Lần 2: sử dụng shellcode đơn để exploit dịch vụ.  
🡺 bro cảnh báo có shellcode

Lần 3: Encode shellcode để bypass các signatures shellcode của bro,kiểm tra unicorn  
🡺 bro không phát hiện  
🡺 unicorn phát hiện

# V.Tài liệu tham khảo:

<http://www.unicorn-engine.org/>

<https://www.bro.org/documentation/>