Vigilante: End-to-End Containment of Internet Worms

Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowston, Lidong Zhou, Lintao Zhang and Paul Barham
Worm

● Standalone malware that replicates itself in order to spread to other computers
● Exponential growth
  ○ Slammer worm infected 90% of vulnerable hosts in 10 minutes!
● Exploit vulnerabilities in Network-facing software
Worm Containment

- Analyze network traffic
- Generate signatures and block matching traffic
- Block hosts with abnormal behavior

- No vulnerabilities detected at network level?
Vigilante’s end-to-end Architecture

- **Host-based detection**
  - Instrument software to analyze infection attempts

- **Cooperative detection without trust**
  - Detectors generate self-certifying alerts (SCA)
  - Detectors broadcast SCAs

- **Vulnerable hosts generate filters to block infection**

- **Automated process**
Vigilante’s end-to-end Architecture
Self Certifying Alerts

- Verifiable proofs of vulnerability
  - Identify the application and a type of vulnerability
  - Contain the exact steps to compromise the host
  - Contain verification information
- Enable hosts to replay the infection
- Verification has no false positives
SCA Types

- **Arbitrary Execution Control**
  - Jump to arbitrary existing code in a service’s address space
  - Specifies how to jump to an address supplied in a message

- **Arbitrary Code Execution**
  - Code-injection vulnerability
  - Specifies how to execute an arbitrary piece of code supplied in a message
SCA Types

- Arbitrary Function Argument
  - Data-injection vulnerability
  - Specifies how to invoke a specific critical function with an argument supplied in a message
Address of code to execute is contained at this offset within the message.
SCA Verification

1. SCA
2. SCA Verifier
3. Verification Manager
4. Virtual Machine
5. Host

Vulnerable Service
Verified
Alert Generation

- Log messages
  - Remove old messages and messages in generated SCAs
- If the engine detects an infection attempt, search the log and generate candidate SCAs
- SCAs that get verified are distributed to the vulnerable hosts
Non-Executable Pages

- Use NX protection on stack and heap pages to detect code-injection attacks
- Search messages for the address or the code that caused the exception
- Use a message as SCA
- Keep adding messages until the SCA is verified
Dynamic Dataflow Analysis

- Track the flow of data received in certain input operations
- This data is marked dirty
- If dirty data is about to be loaded into the program counter, signal an attempt for Arbitrary Execution Control
Dynamic Dataflow Analysis

- If dirty data is about to be executed, signal an attempt for Arbitrary Code Execution
- If an argument to a critical function is dirty, signal an Arbitrary Function Argument alert
SCA Distribution

- Pastry overlay
  - Akamai-like overlay with added security
  - Detectors flood SCAs over overlay links

- DoS prevention
  - Per-link rate limits
  - Per-hop filtering and verification
  - Controlled disclosure of overlay membership
Automatic Filter Generation

- Generate filters by analyzing the execution path followed when the messages in the SCA are replayed.
- Apply dynamic data and control flow analysis to determine the execution path that exploits the vulnerability.
Automatic Filter Generation

Filter Condition: \(\text{msg}[0] + 0x10 = 0x31\)
Automatic Filter Generation

```assembly
mov esi, msg          // move address of message into esi
xor eax, eax          // clear eax register
mov al, byte ptr [esi + 0x00] // move first byte into al
push eax
push 0x10
pop ebx
pop eax
add al, bl            // add 0x10 to al
push eax
push 0x31
pop ebx
pop eax
cmp eax, ebx          // compare with 0x31
jne do_not_drop       // if not equal, do not drop msg
```
The number of instructions executed is larger and the engine has to dynamically translate a number of libraries loaded during the worm attack.
SCA Verification Time

![Bar chart showing SCA verification time for different vulnerabilities: Slammer (10 ms), Blaster (18 ms), and CodeRed (75 ms).]
Filter Generation Time

![Bar chart showing filter generation times for Slammer, Blaster, and CodeRed]
Worm Containment

- Simulation
- Total population of 500,000 hosts
- S of the hosts are susceptible to the attack
- A fraction p of the S hosts are detectors
Worm Containment

(a) Slammer
S = 75.000

(b) CodeRed
S = 360.000

(c) Blaster
S = 500.000
Filter Overhead

![Graph showing filter overhead for SQL, RPC, and IIS. The x-axis represents different types of requests, and the y-axis represents overhead in percentage. The graph compares intercepted requests, intercepted plus filtered requests, and intercepted plus filtered plus attack requests. The values for each category are 0.16, 0.2, 0.51, 0.7, 0.76, 0.76, 1.4, 1.92, and 2.07 respectively.]
End-to-End Experiment

- Five-host Vigilante network
- Host number 1 is a detector
- SCA has to reach vulnerable host number 5
- Time from worm probe reaching 1 till 5 verifies the SCA
  - Slamer: 79ms
  - Blaster: 305ms
  - CodeRed: 3044ms
Conclusion

- Analyzing network traffic is not fast or accurate enough to contain a worm attack.
- Vigilante can contain worms automatically:
  - Requires no prior knowledge of vulnerabilities
  - Fast
  - No false positives
  - SCA enables cooperation across hosts that do not trust each other.