Vigilante: End-to-End Containment of Internet Worms

Abstract
Worm containment must be automatic because worms can spread too fast for humans to respond. Recent work has proposed network-level techniques to automate worm containment; these techniques have limitations because there is no information about the vulnerabilities exploited by worms at the network level. The authors propose Vigilante, a new end-to-end approach to contain worms automatically that addresses these limitations. Vigilante relies on collaborative worm detection at end hosts, but does not require hosts to trust each other. Hosts run instrumented software to detect worms and broadcast self-certifying alerts (SCAs) upon worm detection. SCAs are proofs of vulnerability that can be inexpensively verified by any vulnerable host. When hosts receive an SCA, they generate filters that block infection by analyzing the SCA-guided execution of the vulnerable software. Vigilante can automatically contain fast-spreading worms that exploit unknown vulnerabilities without blocking innocuous traffic.

Introduction
A Worm is a standalone malware that replicates itself in order to spread to other computers. They spread too fast for humans to respond, i.e., the Slammer worm infected 90% of vulnerable hosts in only 10 minutes. This paper proposes a new system called Vigilante which use an end-to-end approach to contain worms automatically. Vigilante introduce the concept of a self-certifying alert (SCA), which is a machine-verifiable proof of a vulnerability. SCAs remove the need for trust between hosts.

The key contributions of this paper are:
1. The concept of SCAs and the end-to-end automatic worm containment architecture it enables
2. Mechanisms to generate, verify, and distribute SCAs automatically
3. An automatic mechanism to generate host-based filters that block worm traffic

Vigilante’s end-to-end architecture
In Vigilante, hosts detect worms by instrumenting network-facing services to analyze infection attempts. Detectors, use this analysis to generate self-certifying alerts (SCA) automatically and they broadcast these SCAs to other hosts. Vulnerable hosts generate filters to block the infection.

Self-Certifying Alerts (SCAs)
They developed three self-certifying alert types with same format, that cover the most common vulnerabilities:

1. **Arbitrary Execution Control alerts (AEC)**
   Identify vulnerabilities that allow worms to redirect execution to arbitrary existing code in a service’s address space

2. **Arbitrary Code Execution alerts (ACE)**
   Describe code-injection vulnerabilities and specifies how to execute an arbitrary piece of code supplied in a message

3. **Arbitrary Function Argument alerts (AFA)**
   Identify data-injection vulnerability and specifies how to invoke a specific critical function with an argument supplied in a message
Alert Verification
Each host runs a VM with a verification manager.

SCA verification procedure:
Step 1: The SCA Verifier receives an SCA for verification.
Step 2: Verification manager uses the data in the SCA to identify the vulnerable service.
Step 3: Then it modifies the sequence of messages in the SCA to trigger execution of Verified when the messages are send to the vulnerable services.
Step 4: If Verified is executed, the Verification Manager signals SUCCESS to the SCA Verifier. Otherwise the SCA Verifier declares failure after timeout. Moreover, Verified load a library and binary rewrite critical functions (e.g, exec syscall).

Vigilante’s verification procedure has three important properties:
- Is fast
- Is simple and generic
- Has no false positives (only some false negatives)

Alert Generation
Hosts generate Alerts when they detect an inflection attempt by a worm. To generate SCA’s, hosts log messages and the networking endpoints where they are received during service execution. If the engine detects an infection attempt, search the log and generate candidate SCAs and runs the verification procedure for each candidate. They use two different Detection Engines:

1. Non-executable pages
   The first detection engine uses non-execute protection on stack and heap pages to detect and prevent code injection attacks. When the worm attempts to execute code in a protected page, an exception is thrown. The detector catches the exception and then tries to generate a candidate SCA.

2. Dynamic dataflow Analysis
   Dynamic dataflow analysis is a generic detection engine that can be used to generate the tree types of Vigilante’s alerts. The idea is to track the flow of data received in certain input operations, for example, data received from network connections. This data and any data derived from it is marked dirty. The engine blocks dangerous uses of dirty data and signals attempts to exploit vulnerabilities:
   a) If dirty data is about to be loaded into the program counter, it signals an attempt to exploit an arbitrary execution control vulnerability.
   b) If dirty data is about to be executed, it signals an attempt to exploit an arbitrary code execution vulnerability.
   c) If a critical argument to a critical function is dirty, it signals an attempt to exploit an arbitrary function argument vulnerability.

Alert Distribution
After generating an SCA, a detector broadcasts it to other hosts over a secure Pastry overlay. This allows other hosts to protect themselves if they run a program with the vulnerability in the SCA. Vigilante uses flooding to broadcast SCAs to all the hosts in the overlay (each host sends the SCA to all its overlay neighbors). The secure overlay also includes defenses against active attacks and prevents attackers from manipulating the overlay topology by enforcing strong constraints on the hostIDs of hosts that can be overlay neighbors.
Automatic Filter Generation
Vigilante uses host-based filters to block worm traffic before it is delivered to the vulnerable service. These filters are unlikely to affect the correct behavior of the service because they do not change the vulnerable service and they allow the service to continue running under attack.
Before the host attempts to generate a filter, it verifies the SCA to prevent false positives. Hosts generate filters automatically by analyzing the execution path followed when the messages in the SCA are replayed. They use a form of dynamic data and control flow analysis that finds the conditions on the messages in the SCA that determine the execution path that exploits the vulnerability.

Evaluation
They use three real worms to evaluate the Vigilante: Slammer, Blaster and CodeRed.

1. Alert Generation
They measure the time to generate SCAs filters with both detectors (NX pages and Dynamic dataflow). The results indicates that both detectors generates SCAs fast. The NX detector performs best because its instrumentation is less intrusive, but it is less general. For both Slammer and Blaster, the dynamic dataflow detector is able to generate the SCA in under 210 ms and it takes just over 2.6 s for CodeRed. Additionally, they measure the size of the SCAs and show that is small and it is mostly determined by the size of the worm probe messages.

2. Alert Verification
Then they measure the average time in ms to verify each SCA. They show that the verification is fast because they keep a VM running, ready to verify SCAs when they arrive and all the necessary code for the verification loaded. The verification VM has low overhead (< 1% of CPU and 84MB Memory)

3. Alert Distribution - Containment of real Worms
The containment of the 3 most popular worms is performed using a simulator. They show that a small fraction of detectors (p = 0.001) is enough to contain the worm infection to less than 5% of the vulnerable population, even under DoS attacks. The Vigilante overlay is extremely effective in disseminating SCAs: once a detector is probed, it takes approximately 2.5 seconds (about 5 overlay hops) to reach almost all the vulnerable hosts. Moreover, the simulation indicates that DoS attacks appear more damaging in configurations where Vigilante is less effective because the significance of DoS attacks hinges directly on the number of infected hosts. Also as expected, Vigilante is increasingly vulnerable to DoS attacks as the verification time increases

4. Filters
Afterwards, they measure the time to generate a filter from an SCA that has already been verified. In all cases, filter generation is fast and the generation filters are effective, without false positives. They also measure the performance overhead introduced by deployed filters and they prove that the overhead is very low in all cases.

Conclusions
Vigilante adopts an end-to-end approach to automate worm containment and introduces the fundamental concept of a self-certifying alert that enables a largescale cooperative architecture to detect worms and to propagate alerts. Self-certifying alerts remove the need to trust detectors and provide a common language to describe vulnerabilities and a common mechanism to verify alerts. Vigilante also introduces a new mechanism to generate host-based filters automatically by performing dynamic data and control flow analysis of attempts to infect programs. These filters can block worms with no false positives and they are effective at containing worms that exploit a large class of vulnerabilities. Finally, Vigilante can contain worms that propagate faster than Slammer even when only a small fraction of hosts can detect the worm.