Summary

Zi Yan

1 Practical Network Support for IP Traceback

We define the approximate trace- back problem as finding a candidate attack path for each attacker that contains the true attack path as a suffix. We call this the valid suffix of the candidate path.

All marking algorithms have two components: a marking procedure executed by routers in the network and a path reconstruction procedure implemented by the victim.

1.1 Basic marking algorithms

- Node append: for any router R, append R to each packet w

  Limitation: 1)high route overhead to append in flight, 2)length of the path is unknown, not enough space for record.

- Node sampling: write router's address into a packet with probability p, only one record will remain. Because the router near the victim will have a bigger chance to mark a packet, therefore sorting the routers by number of marked packets will tend to give out the path.

  Limitations: 1)it is a slow process to reconstruct the path, and a large number of packets are needed, for far away routers have a less chance to mark a packet. 2)not robust against multiple attacks.

- Edge sampling: two static fields reserved for start and end. If the router marks a packet, start is written and distance is set to 0, and if the router chooses not to mark, write R to end when distance is 0, otherwise increase distance by 1. Distance prevent attackers from faking packets, if only a single attacker is there. Because the suffix, an attacker should forge a packet with longer or equal length of true attack path, which is not possible.

1.2 Encoding Issues

To save per-packet space.

- XOR the start and end IP addresses. Reconstruction will begin with the edge from the router adjacent to victim to the victim. A hop further router can be found by XOR again with a hop nearer router.

- only store part of bits of IP address and the offset in the IP address of a route when marking. Cumulatively, the IP address will be recovered.
- Because part of IP cannot be unique, multiple attacks may have a victim gets multiple edge fragments with the same offset and distance. combine IP and its hash value bit-interleavingly, and pick a fragment of this combination to mark, then the following router will XOR at the same offset.

1.3 Reuse IP identification field

2 A Key-Management Scheme for Distributed Sensor Networks

Choose part of keys in a key pool for each node. Shared keys can make two sensors connected. Only need to make all sensors connected.

2.1 Constraints

- Communication security constraints: encryption and decryption consume more power, if use asymmetric algorithm, than that uses symmetric one. Transmission consumes more as well.
- Key management constraints: trusted third party is impractical. Single mission-key is unsafe, if one sensor is compromised. Pair-wise keys take too much space, and adding/removing is expensive.

2.2 Key distribution

- key pre-distribution: 1)generate a large pool of keys and their key identifiers, 2) randomly pick k keys out, 3) loading the key ring into the memory of each sensor, 4) save key identifiers of a key ring and associated sensor identifier on a trusted controller node, 5) for each node, loading i-th controller node with the key shared with that node.
- shared-key discovery: find shared keys from all neighbours by exchanging the key identifiers. Then the topology of all sensors can be established by the connections between sensors.
- path-key establishment: path-key from the k keys for two sensors that are not directly connected.

2.3 Revocation

Whenever a sensor is compromised, all keys in its key ring needs to be revoked. A controller node broadcasts list of revoked key identifiers. The list is signed by K^{ci} which is shared by the i-th controller with each sensor at key pre-distribution phase.

After revocation, shared-key discovery and path-key establishment are needed for the disconnection due to the revoked keys.

Re-keying is like revoking a key from oneself. Shared-key discovery and path-key establishment are needed.

3 SPINS: Security Protocols for Sensor Networks

We present a suite of security protocols optimized for sensor networks: SPINS. SPINS has two secure building blocks: SNEP and μTESLA. SNEP includes: data confidentiality, two-party data authentication, and evidence of data freshness. μTESLA provides authenticated broadcast for severely resource-constrained environments.

3.1 Communication architecture

The current prototype consists of nodes, small battery powered devices, that communicate with a more powerful base station, which in turn is connected to an outside net- work. Base station: the root of a routing forest consisted of sensor nodes.

Communication: 1) node to base station, 2) base station to node, 3) base station to all nodes

3.2 SPINS

- SNEP: data confidentiality, two-party data authentication, integrity, and freshness.
- μTESLA: authentication for data broadcast.

3.2.1 SNEP

Semantic security: 1)sender precedes the message with a random bit string. 2)Two counters shared by the parties (one for each direction of communication).

Use a message authentication code (MAC) to achieve two-party authentication and data integrity.

How to get the keys from a master secret key $\mathcal{X}_{AB}$: To use the pseudo-random function F: encryption keys $K_{AB} = F_{\mathcal{X}}(1)$ and $K_{BA} = F_{\mathcal{X}}(3)$ for each direction of communication, and MAC keys $K'_{AB} = F_{\mathcal{X}}(2)$ and $K'_{BA} = F_{\mathcal{X}}(4)$ for each direction of communication.

The complete message that A sends to B is:

$$A \to B : \{D\}_{<K_{AB},C_A>}, \mathrm{MAC}(K'_{AB}, C_A || \{D\}_{<K_{AB},C_A>}) \tag{1}$$

where D is data, C is counter.

- Semantic security. The counter makes the same message different each time.

- Data authentication. MAC verifies correctly.
- Replay protection. The counter value in the MAC increment prevents replay.
- Weak freshness. A verified message will let the receiver know that the message is sent after previous verified message, because of the increment of the counter value.
- Low communication overhead. The counter is kept at each point without being sent in each message.

Use nonce to get strong freshness, by means of sending request with a nonce, and replying response with the MAC including the nonce.

Initialize a counter exchange:

$$\begin{aligned}
A \to B :& C_A, \\
B \to B :& C_B, \mathrm{MAC}(K'_{BA}, C_A || C_B) \\
A \to B :& \mathrm{MAC}(K'_{AB}, C_A || C_B)
\end{aligned}$$

Request current counter:

$$\begin{aligned}
A \to B :& N_A, \\
B \to A :& C_B, \mathrm{MAC}(K'_{BA}, N_A || C_B)
\end{aligned}$$

To prevent DoS, when bogus messages are got many times, the nodes can send the counter with messages. Another method is to attach another short MAC to the message that does not depend on the counter.

3.2.2 μTESLA

Introduce asymmetry through a delayed disclosure of symmetric keys. Beforehand, we need loose time synchronization. First, base station broadcasts a key that is secret at that time with MAC. Second, at the time of disclosure, base station broadcasts the verification key.

$K_i = F(K_{i+1})$, F is one-way function. Use newer key to verify the old keys. Once one key is disclosed, all keys precede it can be got by applying F on it.

μTESLA has multiple phases:

- sender setup. To generate a one-way key chain of length n, the last key K_n, and computer all previous keys by using one-way function F.
- sending authenticated packets. Uniformed time intervals. Send message in time interval i with K_i. In time interval $(i + \delta)$, reveal the key K_i.
- bootstrapping new receivers. Verify K_{i+1} by $K_i = F(K_{i+1})$
- authenticating packets. Loose time sync for that the sender did not yet disclose the key that was used to compute the MAC of an incoming packet. A new key K_i can be authenticated with K_v by using $K_v = F^{i-v}(K_i)$.

- Nodes broadcasting authenticated data. Memory(store key chain) and energy(compute all keys from K_n) limit a node broadcasting authenticated data. Solution: 1) base station broadcasts instead of the node, 2) node broadcasts the data, but base station keeps one-way key chain and sends keys to the broadcasting node as needed.

4 A Crawler-based Study of Spyware on the Web

An automated solution to three problems:

- Finding executables. 1) *Content-type* HTTP header, 2) URL contained an executable extension, 3) archives, to be extracted, 4) JavaScript, find URL in script
- Running executables within a VM. simulate common user interaction, click next button, fill in user info.
- Analyzing the installed executable. Run anti-spyware tool in VM.

Drive-by Downloads: A drive-by download attack occurs when a victim visits a Web page that contains malicious content. To detect violation of the sandbox of an unmodified browser.

5 Swift: A Fast Dynamic Packet Filter

BPF: compilation, userkernel copying, and security checking.

The primary objective of Swift is to achieve low filter update latency. We attempt to avoid filter re-compilation and optimization, allow "in-place filter updating, and eliminate security checking.

ISA design can avoid compilation and security checking.

Two design choices are made to enable in-place filter modification:

- fixing instruction length. avoid the need to shift instructions on instruction replacement
- removing filter optimization. updates can be applied directly, and only updated part is copied from userspace to kernel. Use hierarchical execution optimization instead.

Hierarchical execution optimization: Reuse existing instructions as parents. Because new primitives usually for the same host but different ports or the same protocol but different hosts. First evaluated parent Pass, if succeeds, just halt, but if failed, execute child passes without re-executing any copied instructions from parent.

A Pass: consisted of a series of instructions connected by logic "AND". Those Passes are used independently and combined by logic "OR". If all evaluations of One Pass are "true", the packet is copied to userspace. Otherwise, it will be evaluated by rest of Passes, and will be dropped if failed all Passes.

6 Packet Vaccine: Black-box Exploit Detection and Signature Generation

- white-box: need source code
- black-box: do not monitor a program's execution flow
- grey-box: no source code, but monitor execution flow

Rather than using expensive dataflow tracking, it detects and analyzes an exploit using the outputs of a vulnerable program.

A key step in most exploits is to inject a jump address to redirect the control flow of a vulnerable program. Our approach is to check every 4-byte sequence (32-bit system) or 8-byte sequence (64-bit system) in a packet's application payload, and then randomize those which fall in the address range of the potential jump targets in a protected program.

6.1 Address Range

obtain a process's virtual memory layout.

6.2 Vaccine Generation Algorithm

- target address set contains stack address range, heap address range, and address ranges of other objects, like global library functions.
- aggregate the application payloads of the packets in one session into a dataflow. Find all byte sequence in target address set.
- for each address found above, replace its most significant byte with a byte randomly picked from a scrambler set R to get a new dataflow.
- construct vaccine packets with new dataflow.

R should not contain symbols which could interrupt a protocol proceeding, like change "GET" in HTTP, and make the byte sequence go out of memory layout.

Exploit semantics should be preserved, and only the jump address should be changed.

6.3 Exploit Detection and Diagnosis

To find exact address causes the segment fault, we can compare all scrambled address with the exception log which includes the address incur a exception. Again, we can change the found address, and exploit the program again to see whether the program still throws exception at the same point.

The CR2 and EIP registers are used to get exploit attempts.

6.4 Signature Generation

Use a known exploit as a template.

6.5 Limitations

1) may destroy exploit semantics while working on binary protocols. 2) cannot work on packets with encrypted payload or checksum. 3) signature have limited expression of exploit conditions.

6.6 Architecture to protect Internet

The found exploit signature will be added into packet filter, so an exploit will be caught at very first time, then be dropped by packet filter.

7 Phinding Phish: Evaluating Anti-Phishing Tools

Using our automated testing system, we were able to test how each of 10 tools responded to a set of URLs multiple times over a 24 hour period, allowing us to observe the effect of blacklist updates and of phishing sites being taken down.

7.1 Tool Exploits

- Content Distribution Networks: Making most blacklist-based tools not work, but SpoofGuard work better since it does not depend on URLs but a non-standard port on which website is on.
- Page Load Attack: A extreme long time to load a page will prevent tools which examine the content of a page, like SpoofGuard, from warning the user with red but not just yellow.

8 Filtering Spam with Behavioral Blacklisting

8.1 IP-Based Blacklist

- Completeness: not all emails to spam traps are blacklisted, false positive spamming emails do not get blacklisted in a short time.
- Responsiveness: taking too long to blacklist a spammer.

8.2 Content-based

Picture, pdf, or other non text attachments cannot be identified easily.

8.3 Clustering algorithm

First clustering, then classification. According to sending IP, received domains, and time. Time axis will be collapsed.

For classification, a recent behavior of an IP will be calculated a score, which is the maximum value among all clusters similarity.

Use existing blacklist to bootstrap.

8.4 Design

SpamTrackers clustering algorithms rely on the assumption that the set of domains that each spammer targets is often more stable than the IP addresses of machines that the spammer uses to send the mail.

- Clustering: 1) initial "seed list" of bad IP, 2) behavior patterns for those IP.
- Classification: give vector r for one IP behavior, return score $S(r)$.
- Tracking Changes in Sending Patterns: In practice, re-clustering, if behavior cannot be classified, or say, map to any existing clusters. In paper, re-clustering in a fixed time interval.

8.5 Others

Cannot find a single threshold to split spamming emails from normal ones.

Can be improved by adding more feature for clustering, or give different weight for different domains.

Incorporated with existing filtering systems or put in everywhere connected to the Internet.

Only cluster rows are sent to compress data, and achieve better reliability with replication and anycast.

Get IP behavior info from trusted source with secure channels. Or adjust clustering time window.

SpamTracker does better in across domain behavior analysis.