

# Research Statement

Nickolai Zeldovich

My research interests are in building secure computer systems. Security problems are everywhere, from desktop computers running spyware, to server compromises revealing millions of social security and credit card numbers, to computer worms bringing down safety systems at nuclear power plants. Lack of security also hampers innovation, as many novel applications are not deployed due to security concerns.

A key problem is that overall security often depends on the correctness of the entire system, which is usually too complex to get right. Most existing approaches to security, such as firewalls or virus scanners, focus on specific aspects of security without considering the system as a whole, and do not qualitatively improve security. My research takes a different approach: it strives to find new system designs in which security largely depends on small parts of the overall system. In most systems, low-level abstractions such as hardware tend to be thoroughly tested and verified. An ideal system would directly leverage these same abstractions to enforce overall application security, such as that of a web server, without unduly increasing the complexity of the abstractions.

Enforcing all security at a low level is difficult because each level of abstraction in a system introduces a *semantic gap* in security, meaning that it is hard to express high-level security policy in terms of lower-level security mechanisms. Instead, most systems implement and enforce different security policies at different levels, and security depends on correctness at every one of these levels. For example, consider an Apache web server running on Linux. Linux provides a security mechanism based on user IDs, which works well for users logging in over SSH. However, Apache has its own security policy, specifying how HTTP users should be authenticated and what web pages and server-side scripts they can access. The Linux user ID mechanism is nearly useless for enforcing such a policy, so Apache implements and enforces its own security. Similarly, the page table mechanism in hardware only protects the kernel from user accesses, and the kernel must enforce the user ID mechanism on its own. The result is that neither secure hardware nor a secure Linux kernel is sufficient for enforcing the web server's security policy.

My research focuses on closing this semantic gap, so that high-level security policies can rest primarily on lower-level security mechanisms. This allows security to cut through complex layers such as application libraries and the kernel, which no longer need to re-implement their own security mechanisms from scratch. The challenge is to design security mechanisms that are *simple* enough to enforce at a low level, such as in a small kernel or in hardware, yet *expressive* enough to let applications specify their high-level security policies.

Of course, building operating systems and hardware around application-specific mechanisms is not a practical solution. Instead, these mechanisms should be flexible enough to support a wide range of applications. I have found that one indicator of an inflexible design is the use of privileged mechanisms for system management, which often indicates a deficiency in the general-purpose mechanisms available to unprivileged applications. For instance, in Unix, only root can set user IDs, which makes it difficult for Apache to use the same user ID mechanism to isolate different HTTP users.

On the other hand, *egalitarian* mechanisms—ones that have no pre-defined privilege levels to determine what kinds of operations can be accessed—tend to be naturally flexible. In my research, I have developed egalitarian operating system security mechanisms that can express security policies of existing systems such as Unix, and at the same time allow applications with different security requirements to implement their own policies using the same mechanisms. Egalitarian mechanisms also make it possible to reuse traditionally privileged system code, such as user authentication, in a less privileged context like a web server, since by design, all operations will still be accessible.

Building a system around a new set of security mechanisms requires rethinking the design at many levels of abstraction, and I enjoy working with all aspects and layers of a system in my research. Making these systems practical also requires building and using full-scale prototypes to understand their limitations and further refine their design. To this end, I have built a fully functional, self-hosting operating system called HiStar that provides novel security properties to applications, developed a decentralized network protocol

called DStar for building secure distributed systems, and co-designed a modified SPARC processor, Loki, running on an FPGA, which provides strong application security guarantees. These systems are described below in more detail.

**Operating system security.** HiStar [1] is an operating system designed around *information flow control* as the one and only security mechanism. Applications in HiStar specify security policies for *data* by controlling how information can propagate through the system. Many high-level security policies can be expressed in terms of information flow, such as “my financial details cannot be sent over the network,” or “a user’s private data can only be sent to that user’s browser.” HiStar’s information flow control mechanism also makes it possible to express access control policies, which can prevent sensitive data from being read in the first place; for example, traditional Unix file read and write permissions correspond to information flowing from or to the file. Finally, information flow control allows reasoning about causality by controlling communication between different parts of the system. As a result, small amounts of trusted code can enforce complex application security policies through interposition.

HiStar’s well-defined information flow control mechanism is simple enough to be enforced by a small kernel of under 20,000 lines of code. To provide strong security guarantees, HiStar breaks down traditional OS abstractions into six kernel object types, along with a small number of operations that act on one or two objects at a time and precisely define where information is flowing from and to. HiStar’s egalitarian kernel interface has no concept of superuser and allows anyone to specify information flow restrictions. To make it possible to manage a system without superuser privileges, HiStar separately controls access to data, such as private user files, and access to resources, such as disk space. Granting the administrator access to the top of the resource hierarchy allows the administrator to control the system’s resources without having any inherent data access privileges.

Using these kernel objects, HiStar implements a self-hosting Unix environment in an untrusted user-level library. However, applications can also bypass the Unix library to implement novel functionality, such as a web server whose security is largely enforced by the kernel, or an entirely untrusted login process in which users can provide their own password-checking agents. HiStar’s egalitarian mechanism also enables modularity: code initially designed for authenticating Unix users

during login was reused without modification by the HiStar web server for authenticating HTTP users.

**Security in distributed systems.** While HiStar enforces security on a single machine, DStar [2] can enforce similar information flow security policies across multiple machines. Distributed systems are often composed of partially trusted machines, and the key to security is in the trust relations between them. To this end, DStar provides a fully decentralized, egalitarian mechanism that gives applications full control over what machines are trusted to handle which parts of the application’s data. By using DStar, HiStar’s web server scales to multiple machines without fully trusting any of them.

The main idea behind DStar is self-certifying information flow policies, which include the data owner’s public key in the name of each policy that applies to that data. The resulting network protocol allows each independent machine to verify trust relations, and separates the specification of trust from the enforcement mechanism. Applications sign trust relations with their private keys, and DStar enforces these trust relations by verifying signatures.

**Hardware security.** HiStar’s information flow control mechanism is simple enough to be partially enforced in hardware, which is an attractive proposition given hardware’s strong track record of thorough testing. The Loki hardware architecture [3] provides a mechanism for associating security *tags* directly with physical memory words. Tags correspond to security policies for memory, and the privileges of a process are defined in terms of the tags it can read or write. Loki’s egalitarian mechanism treats all tag values uniformly, and it is the job of a small *security monitor*, running underneath the HiStar kernel, to translate information flow restrictions specified by applications into tag values and manage the set of accessible tags for each process. As a result, hardware can directly enforce the security of applications such as the HiStar web server.

With Loki, hardware enforces access control according to the application’s information flow policy, and the kernel’s job becomes minimizing the bandwidth of covert information flow channels. As a result, an attacker who compromises the kernel gains access to high-bandwidth covert channels, but cannot bypass the hardware access control mechanism, which enforces policies such as file read and write permissions. Loki’s security monitor is half the size of the HiStar kernel, since it does not try to minimize covert channels or manage devices and page tables. At the same time, Loki adds minimal hardware complexity, and our

FPGA prototype runs HiStar at the same clock speed of 65 MHz as the original processor that we modified.

**Past work.** My previous research explored the role of virtual machines in addressing the problems of system management, security, and mobility [4]. Encapsulating all applications and user state in virtual machines and making virtual machines the unit of management reduces most management tasks into simple operations on virtual disk images. A specialized configuration language further simplifies deployment and management of entire networks of virtual machines [5]. I co-founded a company called moka5 to commercialize these research ideas and prototypes in the context of desktop management.

**Future directions.** I have a strong interest in continuing research on building secure systems, and see a number of possible directions in which to proceed.

I am interested in how hardware mechanisms can improve security. Hardware resources are becoming abundant, with many-core processors appearing in the near future, and I want to explore OS mechanisms that trade off resources for security. For instance, a separate core could be used to execute code handling a sensitive private key in strong isolation. Other upcoming hardware mechanisms hold promise for security as well, such as I/O virtualization support, which can help move complex device drivers out of the fully-trusted kernel, greatly improving overall security. At the same time, complex hardware features like hyperthreading and shared caches often introduce covert channels, making it difficult to control information flow. I am interested in developing mechanisms that can better control the degree of resource sharing to give applications more control over their security; related hardware mechanisms have been proposed for deterministic performance and quality-of-service.

Network security may also benefit from simpler security mechanisms. Existing mechanisms, such as firewalls and intrusion detection systems, are based on complex heuristics that cannot provide strong security guarantees. If applications could specify their security requirements to the network through a simple mechanism, firewalls could in turn provide higher-level security guarantees. A better mechanism to control network resource allocation can also solve a number of security and reliability problems, such as covert channels and denial-of-service attacks. Indeed, denial-of-service attacks can be viewed as a special case of covert channels, where a flood of low-integrity packets interferes with high-integrity packets of an important application.

Unlike the subjective and often-changing security mechanisms of traditional systems like Unix, well-defined information flow control mechanisms are particularly well suited to formal verification, since any violation can be objectively proven by example. Model-checking can be used to verify whether a design is secure, and preliminary work has proven the security of a subset of the HiStar kernel interface. HiStar's small kernel is also amenable to verifying the implementation for correctness. Preliminary work in this area is focusing on test coverage using static analysis and symbolic execution, but proving security is the long-term goal.

Security mechanisms must be usable by application developers, and I am interested in exploring how programming languages can make it easier to develop secure applications and express security policies. Applications running on mobile devices, such as cell phones, pose yet another set of challenges: what mechanisms are needed to enforce security in a network of wireless devices with intermittent connectivity, or to prevent malicious code from draining battery power? I also want to explore system designs in which high-level application security can primarily rest on the correctness of cryptographic mechanisms.

Finally, security is only a step toward novel functionality. Better system security will enable innovative applications in areas currently limited by security and isolation concerns, with finance, health care, and air traffic control being just a few examples. Exploring such applications would be fruitful research in itself.

## REFERENCES

- [1] Nikolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in HiStar. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 263–278, Seattle, WA, November 2006.
- [2] Nikolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. Securing distributed systems with information flow control. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, April 2008. To appear.
- [3] Hari Kannan, Nikolai Zeldovich, Michael Dalton, and Christos Kozyrakis. Architectural support for minimizing trusted code. In submission, November 2007.
- [4] Ramesh Chandra, Nikolai Zeldovich, Constantine Sapuntzakis, and Monica Lam. The Collective: A cache-based system management architecture. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 259–272, Boston, MA, May 2005.
- [5] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nikolai Zeldovich, Jim Chow, Jim Norris, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th USENIX Large Installation System Administration Conference*, pages 181–194, San Diego, CA, October 2003.