Testimony before the US House Committee on Science, Space, and Technology.
November 19, 2013
Rayburn House Office Building, Room 2318
Aviel D. Rubin, Ph.D.

My name is Avi Rubin, and I am Professor of Computer Science at Johns Hopkins University and a former Fulbright Scholar. I am also Technical Director of the Information Security Institute and Director of the Health and Medical Security (HMS) Lab at Johns Hopkins. I have been working in IT Security since 1992, and my Ph.D. was in the area of network security and applied cryptography. Before coming to Johns Hopkins almost 11 years ago, I spent 9 years working in the Bell systems research labs on security issues including Web security, data privacy, and general IT security. I am author or co-author of five books on the subject.

I am currently advising six Ph.D. students and over a dozen undergraduates, and my lab is funded by the Office of the National Coordinator for health information technology and by the National Science Foundation. My grants target healthcare IT and electronic medical records security. My sponsored work does not relate in any way to the HealthCare.gov site or any other government system in production.

From 2005 to 2011 I ran a software security consulting company that evaluated the security of systems, including large Web deployments and backend databases.

I have been asked to comment on security issues for large Web installations in general, and to address, specifically, security issues that need to be considered for the HealthCare.gov Web site.

My understanding is that among other things, HealthCare.gov collects some sensitive information from users to assess their eligibility. The site communicates with databases held by the IRS, DHS, and SSA to verify eligibility for federal subsidies and with third-party non-governmental entities like Experian to verify patients' identities. HealthCare.gov does not collect nor store Electronic Medical Records, but it does collect whatever personal information is needed for enrollment. This information, in the wrong hands, could potentially be used for identity theft attacks.

There have been many highly publicized breaches of large online systems where credit card information, social security numbers, and user passwords have been exposed. Some of the more notable ones involved Heartland Payment Systems, TJ Maxx, and most recently Adobe. Rarely does a week go by without a major media story about a new data breach. Anytime valuable, sensitive information is managed through a user-facing Web interface, there is a risk of exposure, and attackers are constantly growing in sophistication, creativity and resources. As one of the largest and most complex undertakings in the online space, HealthCare.gov faces the same security challenges as other online sites such as airline reservation systems, online banks and retailers, and large social media sites.

It has been widely publicized that HealthCare.gov has had a rocky deployment. To a software engineer, this is not surprising. The system is very large, and it interoperates with many different Web sites and back end systems. The success of HealthCare.gov depends not only on the software and servers that run the front end of the site, but also on every one of the organizations with which it shares data on the back end. Furthermore, HealthCare.gov was deployed with a hard deadline for going live, and there were indications that the system was not ready, as the deadline approached.

When software systems run behind schedule, the temptation is to increase the manpower to try to catch up. However, it is a well-known mantra in software engineering that adding people to a late software project is likely to make it later. A famous book by the software pioneer Dr. Fred Brooks titled The Mythical Man-Month captures this idea, and is considered one of the all time classic books on the development and deployment of large software products. Once a project falls behind schedule, sticking to a hard deadline can result in a faulty system that is not properly tested. Furthermore, systems that may appear to work well in the lab, often fail when scaled up to a large number of users in the field. Stress testing a large-scale system requires simulating the actual environment in which the software will run when hundreds of thousands of users simultaneously access it. Such simulations often do not properly test the system under a realistic load.

The issue of scale is an important one. Most large, consumer-facing Web-based rollouts happen in phases. For example when Google introduces a new service, they initially offer it to a select group of users. As bugs are ironed out and problems are resolved, the new functionality is enabled for more users. It is an iterative process, and there are always issues to resolve. One of the biggest mistakes of HealthCare.gov was the decision to roll it out all on one day. That is not the way large systems go live in practice.

One of the basic principles of security is that a system's security is inversely proportional to its complexity; that is, the more complex a system is, the more numerous vulnerabilities in that system will be. In other words, "Keep it simple" is the best advice. When a system must be complex by its nature, such as is the case for HealthCare.gov, then a good way to address security in the design is to focus on well-defined interfaces among components. This is part of building in security from the beginning.

One cannot build a system and add security later any more than you can construct a building and then add the plumbing and duct work afterwards. That said, in practice, software systems evolve, and as a system changes, new security considerations arise. In practice, systems require some post-production "bolting on" of security features and retrofitting security solutions despite any efforts to build security in at the outset. Ongoing vigilance and response are needed to properly maintain a secure Web installation.

I have followed news reports of some security problems with HealthCare.gov. As far as I can tell, so far all of the security problems that have been publicized were easy to fix and have been remedied. Assessing whether there are any deep, architectural security flaws will require an in-depth design review by security specialists. In the meantime, I have several recommendations that I list at the end of this testimony on how to maximize the security of HealthCare.gov.

Maintaining a secure Web site is not easy and requires ongoing maintenance, administration and expertise. That said, there are many Web sites that operate successfully and which have not, to my knowledge, suffered any significant breaches. Given the large number of interoperating systems and the sensitivity of the data that it handles, I classify HealthCare.gov as a high-maintenance system from a security perspective. It cannot be deployed and left alone. High quality system administrators are needed to keep up with software patches from vendors, to respond to incidents, and to monitor the systems for suspicious incidents. A contingency plan should be developed for every conceivable incident, and a reporting system should be put in place so that responses can occur in a timely fashion.

I believe that if security best practices are adhered to, if the system was architected with proper security and well designed interfaces on the back end, and if my recommendations below are followed, that it is possible for a site with the objectives of HealthCare.gov to achieve the same level of security as some of the well-known popular Web sites that people use regularly on the Internet to shop, bank, book travel, keep up with their friends, and otherwise manage their lives. There will always be the potential for security incidents, but the risks can be minimized with proper design, management and administration.

Here are my recommendations for securing HealthCare.gov:

- Outside, independent experts should review the security of the system annually, including design review, code review and red team exercises
- Security reviews should focus on the interfaces among the components and across systems.
- User authentication mechanisms should be reviewed, and two-factor authentication should be employed wherever practical.
- Security reviews should check for known standard vulnerabilities such as SQL injection attacks, sanitization of user inputs, Cross Site Scripting vulnerabilities, and other standard checks.
- Data at rest should be encrypted, and keys should be cleared from memory when they are not in use.
- Implement mandatory incident reporting, even of suspected and unconfirmed incidents, and contingency plans should be designed for conceivable scenarios.

The opinions expressed herein are my own and do not necessarily reflect the views of The Johns Hopkins University. Thank you for the opportunity to testify before this committee, and I look forward to answering your questions.