conference reports

- This issue's reports focus on the 13th USENIX Security Symposium, held in San Diego, California, August 9–13, 2004.
- Our thanks to the scribe coordinator:

  Rik Farrow
- Our thanks to the summarizers:

  Alvin AuYoung
  Eric Cronin
  Marc Dougherty
  Serge Egelman
  Rachel Greenstadt
  Stefan Kelm
  Zhenkai Liang
  Chad Mano
  Nick Smith
  Ashish Raniwala
  Tara Whalen
  Wei Xu

KEYNOTE ADDRESS

Back to the Future

William "Earl" Boebert, Sandia National Laboratory

Summarized by Tara Whalen

Earl Boebert opened his remarks by saying that his views were his own, since "nobody in their right mind would let me speak for them." The wealth of knowledge and insight expressed in his keynote talk demonstrated that nobody in their right mind would ignore his expertise. Boebert stated that a lot has been forgotten about security over the years, so it is necessary for somebody who's been around for a while to speak up.

Boebert said that the way to build buildings that stay up is to look at buildings that fall down. Security experts should not ask, "Why does it work?" but "How does it fail?" What is worrisome is that currently insecurity has been pushed to the end nodes, which is the worst place for it to be. Why did this happen? It was driven by economics: You make money by giving people what they want. Quoting comedian Rich Hall, Boebert said that what Americans wanted was "crap in a hurry," and, apparently, so do computer people. However, an alternative exists: engineering. Software developers used to aspire to well-engineered solutions before the industry was overwhelmed with the "get rich quick" ethos.

Boebert went on to talk about the higher-level goals of engineering: operational and formal assurance. Operational assurance is what you gain from experience: "It hasn't killed anybody yet, so it must be okay." However, to gain operational assurance, you must always carry out your operations in exactly the same way. Instead, you could rely on formal assurance, a structured argument that shows what you can safely do. Boebert then talked about assurance in the context of his experience with Multics.

Multics was developed around principles that supported formal assurance, including unity of mechanism (doing a task all in one place) and separation of policy and mechanism. In addition, one of its design strengths was simplicity. For example, Multics' virtual memory design was useful for what it got rid of: buffers, inodes, puts, and gets, which were replaced by a unified name space. Also, backups were automatic and silent; Boebert used Multics from 1969 to 1988 and never lost a file.

Multics was based on a "large process model" of the movement of an execution point through code modules. This supports assurance arguments about system security. For example, in Multics, access rights were part of the segment descriptor word (used for the page table). This provides a "hardware lemma": You cannot avoid hitting the access bits, because they are integral to forming a hardware address.

Boebert also discussed online threats, using the term "Internet slime" to describe such problems as active content. What you want to have is a system such that the Internet slime cannot under any circumstances get access to the kernel. Following the Multics design, you set up an intermediate module between the slime and the kernel: slime can't call the kernel object directly, but has to go through a "gate" object. The intermediate module performs argument validation, after which it can provide access to the kernel object, which keeps the slime object safely away. In case a really bad slime object tried to fool the kernel object into calling back to the slime, the hardware ring would trap it and stop the disaster from happening.

Other good features of Multics included pointers, single copies of procedures, and heavy use of dynamic linking. A NASA study concluded that you should either change software a little every day or

a lot once a year if you want to have a stable system. Because it supported online software updates, Multics was remarkably stable.

Boebert added that "we had some lousy ideas back then, but in our own defense, we never came up with anything as silly as security as a side effect of copyright enforcement." He concluded with a brief prognosis: "When an old fart comes up here, he's supposed to forecast the future optimistically, so we all go away with a nice feeling." However, he is not optimistic that we will ever see a system as rigorously engineered as Multics: The desire for "crap in a hurry" will prevent this in the foreseeable future, sadly.

One audience member asked how we could bring back the climate in which Multics was engineered. Boebert replied that he would try to reinstate the principles, rather than the system itself. He thinks that it would be nice if somebody would "do one [operating system] right for the sake of doing one right," as a learning experience.

ATTACK CONTAINMENT

Summarized by Stefan Kelm

A Virtual Honeypot Framework

Niels Provos, Google, Inc.

This year's first refereed paper was presented by Niels Provos, the author of honeyd, which he described during this talk. A honeypot can be defined as a computing resource that we explicitly want to have probed or attacked in order to study an attacker and his actions on our system. In terms of their implementation, honeypots can be divided into low-interaction vs. high-interaction honeypots as well as physical vs. virtual honeypots.

honeyd offers a framework for creating low-interaction virtual honeypots. It is able to simulate TCP, UDP, and ICMP packets and to adapt what Niels described as "different operating system personalities." Thus, honeyd simulates the TCP/IP stack behavior of a huge number of today's known OS implementations, thereby hiding the IP stack of the actual system honeyd is running on. This is realized by the so-called "personality engine," one of the core components of honeyd. The personality engine is based on nmap's fingerprint file: Anyone running nmap against a honeyd system will therefore get whatever OS has been configured to show up.

Another core component is the traffic dispatcher: Since honeyd does not itself intercept any network traffic, the traffic needs to be redirected to honeyd. The traffic dispatcher then has to decide how to handle any incoming packets. Since there are several ways to redirect traffic, honeyd is able to simulate not only a single IP address but complete network topologies, consisting of multiple hosts running different operating systems. Interestingly, honeyd seems to do very well performance-wise: On a standard PC, honeyd is able to simulate roughly 2000 TCP connections per second.

Niels went on to describe possible applications, namely, detecting and disabling worms as well as preventing spam. The author concluded that honeyd provides an efficient and scalable framework that deceives fingerprinting. Future work will be to enhance honeyd for further attack detection.

A number of interesting questions were asked by the attendees, two of which focused on honeyd in an IPv6 environment. Niels acknowledged that deploying IPv6 honeypots will likely be more expensive.

For more information, contact niels@google.com or see http://www.citi.umich.edu/u/provos/honeyd/.

Collapsar: A VM-Based Architecture for Network Attack Detention Center

Xuxian Jiang and Dongyan Xu, Purdue University

This talk can be seen as a follow-up to the previous one. Dongyan started by describing some of the problems of current honeypots, namely, lack of expertise, inconsistencies when operating multiple honeypots, and no central management. His solution is called Collapsar, a VM-based architecture which is founded on Lance Spitzner's honeyfarm idea.

Collapsar tries to integrate two goals: implementing a distributed honeypot presence within a network topology, while maintaining a centralized honeypot operation. It consists of three main functional components: (1) the redirector runs in each participating network and captures all traffic to unused IP addresses, which is then redirected to another component; (2) the front end acts as an interface between the redirectors and what is called the "Collapsar center"; (3) the virtual honeypots inside the Collapsar center are implemented as high-interaction honeypots and are therefore able to simulate different operating systems as well as popular services such as Sendmail and Apache. In addition, there are a number of different assurance modules, providing, for example, traffic logging and subsequent data correlation.

Dongyan next described the results of a Collapsar experiment they've been running. The honeypot architecture was deployed in a local environment that consisted of traffic redirected from five different networks with 40 honeypots running within the Collapsar center. During that experiment they were able to do a forensic analysis on incidents such as attacks on Apache or on XP. As to the performance, they measured TCP throughput as well as ICMP latency

for both VMware and UML (user-mode Linux).

The authors observed that Collapsar even allows for more sophisticated analyses, such as stepping-stone identification and detection of network scans.

During the discussion one member of the audience remarked that Collapsar relies on the (bad) principle of "security by obscurity" in that the redirectors need to remain in the "dark IP space" (i.e., hidden from possible attackers) in order to function properly. This was confirmed by Dongyan.

For more information, contact dxu@cs.purdue.edu or see http://www.cs.purdue.edu/homes/jiangx/collapsar.

Very Fast Containment of Scanning Worms

Nicholas Weaver, International Computer Science Institute; Stuart Staniford, Nevis Networks; Vern Paxson, International Computer Science Institute and Lawrence Berkeley National Laboratory

Nicholas Weaver talked about a new and very fast method to scan for kinds of "Internet slime" in order to stop these worms efficiently. Static defenses have been and still are insufficient to stop worms from spreading in networks. The reaction needs to be automatic.

The algorithm Nicholas and his group have developed aims to implement worm containment, i.e., isolating a worm and subsequently stopping it from spreading further. Their work is based on the Threshold Random Walk (TRW), which has been modified for better hardware and software implementation possibilities. The basic idea is to use caches in order to keep track of incoming and outgoing connections; if a particular counter has been reached, connections start being blocked.

One interesting addition to the algorithm seems to be cooperation between said containment devices. If a device is able to use the status of other devices as another detector, the scanning results should be improved. The speaker concluded that not only are they able to enhance worm containment using their implementation, but they also gain a better understanding of particular worm attacks.

For more information, contact nweaver@icsi.berkeley.edu.

RFID: Security and Privacy for Five-Cent Computers

Ari Juels, RSA Labs

Summarized by Ashish Raniwala

Ari Juels, principal research scientist at RSA, gave an excellent presentation on security and privacy issues for RFID. RFID does not refer to any single device but to a spectrum of them, ranging from the basic RFID tag and EZ Pass to mobile phones. The talk focused on the basic RFID tag, which is a passive device that gets all its power from the RFID reader. In contrast to a bar code, RFID does not require line of sight between the tag and the reader. Additionally, RFID provides a unique ID for every object, as compared to bar codes, which only identify the type of object. An RFID has fairly limited memory (hundreds of bits), computational power (thousands of gates), and wireless range (few meters), making it hard to implement any real cryptographic functions or non-static keys.

RFID is already seeing many practical applications, starting from supply-chain visibility to anti-counterfeiting drugs. With push from such industry giants as Wal-Mart and Gillette and the decreasing cost of tags and readers (estimated to be 5 cents/tag and several hundred to several thousand dollars/reader by 2008), RFIDs are expected to become the physical extension of the Internet.

However, there are several security and privacy issues associated with RFID usage. An RFID tag can be clandestinely scanned to get personal information such as items carried on person. Limited computing resources make it hard to enforce strong access control on the RFID. Because of such concerns, RFIDs are already facing strong opposition, not just from customers but from corporations as well. For example, corporations need to worry about espionage and tag counterfeiting.

One approach to address the privacy issues is to "kill" the RFID tag as soon as the customer checks out items from the store. However, RFIDs are much more useful in their "live" state. Ari mentioned several novel applications that require that RFIDs stay alive post-checkout. For example, one can imagine a "smart" closet that can detect what clothes one has and suggest latest styles, or a "smart" medicine cabinet that can aid cognitively impaired patients, or automated sorting of recycled objects.

Ari suggests that it is hard to come up with useful solutions if one assumes the omnipresent oracle adversarial model. One needs to work with more realistic and weaker adversarial assumptions. Ari discussed three different approaches based on his own research. The first approach, termed "minimalist cryptography," is based on the observation that an adversary has to be physically close to the tag to be able to read it, and therefore can query it only a few times in any attack session. One can therefore store multiple pseudonyms on the tag, and return a different one each time the adversary queries. This makes it hard for the adversary to associate any tag with a particular object. The object-pseudonyms association can be known only to a trusted verifier. This approach can be further strengthened by throttling the query rate to prevent rapid, repeated scanning of the tag, and by refreshing the set of pseudonyms using a trusted reader.

The second approach is based on the idea of a "blocker tag," which can simulate all possible tags for any object. The blocker tag exploits the "tree-walking" anti-collision protocol used in ordinary tag-reader communications, and returns both "0" and "1" every time the reader queries for the next bit of the tag. This spams the reader with a large number of tags, protecting the privacy of the person. A blocker tag, however, needs to be selective in its blocking; for instance, it should only block purchased items carried by a shopper, not the unpurchased ones. Juels discussed use of a privacy bit that can be turned on or off based on whether the user wants the item to belong to the privacy zone.

The third approach uses a proxy RFID device, such as a mobile phone. The proxy device acquires the tags of the object and deactivates the original tag. All queries to the tag are then answered by the proxy device while enforcing the desired privacy policies. The proxy device can release the acquired tag and reactivate the original tag when it is about to leave the wireless range of the proxy.

RFIDs have led to a significant privacy debate, but at this point these security and privacy problems are more psychological than technological. Currently, RFID deployers can barely get the technology to work. For instance, UHF tags do not work well when close to the human body. It is hard to distinguish items in one shopper's cart from those in another's, because of the uncontrolled wireless range.

Ari concluded the talk by discussing some of the open issues for research: more realistic adversarial models for evaluating security techniques, and anti-cloning to prevent cloning of RFIDs. Ari pointed to *http://www.rfid-security.com* for more information.

PANEL: CAPTURE THE FLAG

Giovanni Vigna, University of California, Santa Barbara; Marc Dougherty, Northeastern University; Chris Eagle, Naval Postgraduate School; Riley Eller, CoCo Communications Corp.

Summarized by Rachel Greenstadt

The goal of this panel was to explore the utility of Capture the Flag competitions as pedagogical tools. These are competitions in which teams compete to defend their computers/data and attack those of the other team. A scoring bot periodically gauges the progress of the various teams and assigns points.

The panel, moderated by Tina Bird, got off to a rough start as Riley/Caesar—representing the Ghetto Hackers—was delayed. The Ghetto Hackers run the famous Capture the Flag competition at DefCon. He arrived by the end, but the panel began with the other panelists: Chris Eagle, associate chairman of the Naval Postgraduate School and a member of their Capture the Flag team; Marc Dougherty, who runs a Capture the Flag competition at Northeastern; and Giovanni Vigna, an associate professor of CS at UCSB who runs Capture the Flag competitions at his school and participates in the DefCon competition. It turned out Chris represented the winning team at this year's DefCon, and Giovanni, the second place team.

Marc Dougherty got his start in Capture the Flag competitions by participating in the competition at Northeastern and winning by exploiting a trivial weakness in the game setup. He decided to help fix the competition and has been involved in it ever since. He likes Capture the Flag competitions because they give students an opportunity to play with tools they wouldn't get to play with (legitimately) otherwise. At Northeastern, Capture the Flag is all-volunteer, not part of any class, and is all about fun. They run their competition attached to the Internet (as opposed to most other competitions, which have an air gap), but the students are careful. Marc's working on a modular system for running and scoring games so that they are easier to set up and run. He hopes to have it out, and available to the public, by October.

Chris has been involved with Capture the Flag for four years; this year his team won at DefCon. In describing their victory, he explained how the setup of the game can have unintended consequences. This year's DefCon was supposed to be all about offense—85% of the points were gained from offense, only 15% from defense—but Chris acknowledged that Giovanni's team had a better offense than his. However, the way the scoring bot worked was to access each system and then award points based on how many of your own and other teams' tokens you had. Chris's team won by waiting for Giovanni's team to hack a bunch of systems and then hacking theirs, stealing all their tokens and defending them well.

Giovanni concurred with this analysis and also mentioned that DefCon was tricky this year because the VM image of the system they were supposed to attack and protect was Windows, and who knows how to deal with that? Giovanni talked more about running a competition in a university setting. Capture the Flag competitions are a great tool to educate students in security. In this hands-on experience, the students learn a lot about buffer overflows, SQL injection, etc. You can also structure the competition so that you get useful data for your research, which is a nice side effect. The competitions help students develop teamwork and crisis management skills. All the panelists agreed that the most difficult part of running the competi-

tions was developing a good scoring system. A competition requires a lot of time to set up initially, but running it becomes easier after that.

This was Caesar's fourth year involved in running the DefCon Capture the Flag competition. After the Ghetto Hackers won the competition the third time in a row, they decided to run it. This year they changed the system so that the scoring bot monitored who had the tokens of the various teams. Before this, scoring was just based on rooting a box, which is not an interesting measure per se, since it doesn't provide a realistic view of the way information is stolen today (at this point, Tina noted that two of the panelists' eyebrows moved). The competition was easier to prepare this year; they got people with a bunch of drinks coding nonstop for 10 days and it was done. All the panelists praised the way the DefCon qualifying round was run this year. The Ghetto Hackers are organizing a large, Internet-based contest, called Mega Root Fu. The Ghetto Hackers are willing to run competitions in exchange for airline and hotel costs and a weekend beer budget ($200–$300 of alcohol a day).

Fighting Computer Virus Attacks

Peter Szor, Symantec Corporation

Summarized by Wei Xu

Every month, critical vulnerabilities are reported in a wide variety of operating systems and applications. Computer virus attacks are quickly becoming the number one security problem and range from large-scale social engineering attacks to exploiting critical vulnerabilities. In this talk Szor discussed the state of the art in computer viruses and computer virus defense. He presented some promising host-based prevention techniques that can stop entire classes of fast-spreading worms, as well as worms using buffer overflow attacks. He also discussed in-depth worm and exploit analysis.

Szor first presented a short history of self-replicating structures. Modern computers started from von Neumann's architecture, in which programs are stored in primary memory and the CPU executes the instructions one by one. In 1948 von Neumann introduced Kinematon, a self-reproductive machine. Later Ed Fredkin used cellular automata to represent self-reproductive machines. In 1966, warrior programs, Darwin and Core Wars, were devised. In these programs, warriors fight each other, which is very similar to modern worms. John Horton Conway's 1970 Game of Life is unusual. The game follows a set of simple rules, but it can produce a remarkably large number of patterns. In the early seventies, self-replicating programs emerged. For instance, in 1971–1972, Creeper was born during the early development of ARPANET. In 1975, the self-replicating version of the Animal game was released, written by John Walker, founder of Autodesk, Inc. In 1982, Skrenta's Elk Cloner, a virus on Apple II, was developed. Peter showed a quick demo of Elk Cloner using an Apple II emulator.

Peter then presented several representative virus code evolution techniques. The complexity of virus threats has increased over time. In 1987, code encryption began to be used to hide the function of the virus code, and in 1990, code permutation was introduced into viruses. The number of permutations of a single virus has increased significantly. For example, in 2000 W32.Ghost had 10 factorial different permutations. Polymorphic viruses, which first appeared in 1991, can create an endless number of new decryptors that use different encryption methods to encrypt the constant part of the virus body. An even more powerful technique used by polymorphic viruses is external polymorphic engines (Polymorphic Engine Object, or MtE). Emulation provides a very good approach to detection of this type of virus.

Metamorphic viruses are more difficult to detect. Here "metamorphic" means "body polymorphic." Unlike other polymorphic viruses, metamorphic viruses do not require a decryptor. Instead, the virus body is "recompiled" in new generations, and the virus code can shrink and expand itself. Source code mutation (encrypted source code compiled into slightly different but equivalent encrypted source code) is usually used in metamorphic viruses (e.g., W32.Apparition). Equivalent instruction sequences are one s uch technique. There are metamorphic viruses for Windows operating systems, including the recent Windows .NET framework (MSIL.Gastropod in 2004). Surprisingly, there are also multi-platform metamorphic viruses, such as {W32, Linux}/Simile.D, which infects both Windows and Linux systems.

There is a class of "undetectable" computer viruses. W95.Zmist is one example. This virus is an entry-point-obscuring virus that is metamorphic. Moreover, the virus randomly uses an additional polymorphic decryptor. Zmist also supports a unique new technique: code integration. The Mistfall engine contained in the virus is capable of decompiling portable executable files to their smallest elements. Zmist first moves code blocks out of the way, inserts itself into the code, regenerates code and data references, including relocation information, and rebuilds the executable.

Next Szor talked about modern worm attacks. Nowadays, we have another important problem: fast-spreading worms. The frequency of worm attacks is increasing, and worms are getting faster than update and patch deployment. Meanwhile, known vulnerabilities are on the rise. There are many

incidents of worms that caused major DoS attacks; for instance, the Blaster worm contributed to a major blackout. Here are several famous worms: Blaster (August 2003), Code Red (July 2001), Nimda (September 2001), Slammer (January 2003).

The Code Red worm is based on a buffer overflow attack on Microsoft Internet Information Server (IIS). The worm uses GET requests to trigger the buffer overflow vulnerabilities in IIS.

The Slammer worm is another buffer overflow–based attack targeting Microsoft SQL Server. The Slammer code is amazingly small, only 376 bytes.

To detect these types of worms, we can use generic IDS signatures characterized by the sizes of buffer and input strings as well as other important attack patterns. Below is a rule to detect Slammer:

```
alert udp any any -> any
1434 (msg: "MS02-039
exploitation"; content:
"|04|"; offset 0; depth:1;
dsize>60)
```

where 60 is calculated by size_of_buffer - (string_size1 - string_size3 - terminating_zero) = 128 - 40 - 27 - 1 = 60.

Szor gave the details of how Blaster works. Basically, Blaster exploits the target on port 135/tcp of Windows XP/2000 and uses the TFTP protocol to upload the worm code onto the victim systems.

Szor then presented various worm-blocking techniques.

In a host-based layered security model, we have layers such as personal firewall, anti-virus protection, host-based IDS, and OS/application layer prevention or access control. The network can also be a part of this model. Outbound SMTP worm blocking is an example of network protection. In this case, emails sent by an SMTP client are first redirected to a scan manager for virus scanning before being passed to the SMTP server. The scan manager can use a NAV client and consists of a decomposer and anti-virus engines.

Digital Immune System can provide worm/virus protections for a large-scale network. In this system, the customer-side AV client communicates with a quarantine server, which itself talks to the vendor-side servers through a customer gateway. The vendor-side servers include automated/manual analysis centers and definition servers. Anti-virus researchers interact with these servers to identify new viruses/worms and provide detection mechanisms.

There are many techniques for blocking buffer overflow attacks: code reviews, security updates (patches), compiler-level extensions (e.g., StackGuard; ProPolice; and Buffer Security Check in MSVC .Net 7.0/7.1), user-mode extensions (e.g., Libsafe), kernel extensions against user-mode overflows (e.g., PAX, SecureStack), and OS built-in protections (e.g., Solaris on SPARC).

An NX (non-executable) page attribute on AMD64, EM64T, and updated Pentium 4 can be used to prevent buffer overflow attacks. When an NX-marked page is executed, an exception is generated. The Data Execution Prevention (DEP) in Windows XP SP2 takes advantage of this feature. Note that there are already viruses that are aware of DEP.

In order to block worms, we can also deploy other techniques such as exception-handling validation, injected code detection (shell-code blocking), and self-sending code blocking (send blocking). These techniques can stop attacks after a buffer overflow or prevent viruses/worms from propagating.

Szor showed demos on blocking shell code in both Windows and UNIX systems. He also showed demos on blocking various worms. It is worth mentioning that the Witty worm made use of exploitations of third-party software (Internet Security Systems' security products).

In the end, Szor pointed out some future threats from viruses/worms.

Worms can do secure updating. For example, the Hybris worm has more than 20 plug-ins, and these plug-ins can be downloaded via secure channels by the worm itself.

Worms can communicate. They can export/import user accounts/passwords, payloads, mutation engines, and exploit modules.

Viruses for mobile phones, smart phones, and pocket PCs will become a huge problem. In fact, there already exist proof-of-concept viruses for these platforms.

Q. Have you seen any viruses on mobile devices that spread by means other than Bluetooth?

A. Haven't seen such viruses. But we expect to see viruses that leave a backdoor on mobile devices and use SMTP to send out emails with attachments or invitation SMS messages to infect other devices.

Q. How would I get access to proof-of-concept viruses?

A. No public accesses. We get them by submissions.

Q. What is the meaning of "proof-of-concept" viruses?

A. These are viruses that do not want to hide themselves.

Q. How can one detect Zmist today?

A. It is hard. But we can detect the virus using a disassembler.

Q. Is there convergence between anti-spam and anti-virus efforts?

A. We have recognized that anti-spam techniques are very important. We are investigating this.

Summarized by Chad Mano

TIED, LibsafePlus: Tools for Runtime Buffer Overflow Protection

Kumar Avijit, Prateek Gupta, and Deepak Gupta, IIT Kanpur

Kumar Avijit presented a two-pronged approach for protecting both new and legacy code from the "menace" called buffer overflow. This is accomplished by using TIED (Type Information Extractor and Depositor) and LibsafePlus.

TIED uses debugging data generated by the compiler to create a data structure containing the size information of all global and automatic buffers. This data structure is inserted into the program binary to be used at runtime. LibsafePlus is an extension of the Libsafe tool which provides more extensive bounds-checking features. LibsafePlus identified all 20 buffer overflow attacks in the Wilander and Kamkar test suite, while Libsafe identified only six.

The overhead associated with these tools is around 10 percent. The code is available at http://www.security.iitk.ac.in/projects/Tied-Libsafeplus.

Privtrans: Automatically Partitioning Programs for Privilege Separation

David Brumley and Dawn Song, Carnegie Mellon University

David Brumley explained that attackers target privileged programs because "it gives them a bigger bang for their buck." Programs can be partitioned into two separate entities to increase protection. This prevents a bug in the unprivileged part of the program from giving access to a privileged area. The privileged program, or monitor, handles sensitive processes, such as opening /etc/passwd. The unprivileged program, or slave, handles all other processes. The important result of partitioning is a small monitor, which means the trusted base is easier to secure.

Partitioning of programs is traditionally done manually. Mr. Brumley presented the first approach to automatic privilege separation. This is accomplished using a tool called Privtrans. Variables and functions in the source code must first be annotated to identify the privileged parts of the program. For the programs tested this took between 20 minutes and two hours. Privtrans takes the annotated source and creates separate monitor and slave source code. Communication between the two programs is implemented with RPC.

Privtrans is able to obtain results similar to manually partitioned code. The associated overhead is reasonable for the increase in software security.

Avfs: An On-Access Anti-Virus File System

Yevgeniy Miretskiy, Abhijith Das, Charles P. Wright, and Erez Zadok, Stony Brook University

Yevgeniy Miretskiy presented Avfs, which is a true on-access virus-scanning file system. In contrast to on-open, on-close, and on-exec scanning, which run in user space, Avfs runs in kernel space and provides incremental file scanning, which prevents infected files from being written to disk.

Avfs is based on the ClamAV virus scanner, which was enhanced and renamed Oyster. The enhancements improve the scalability from a linear function to a logarithmic function. The important features of Avfs are that it detects viruses early to prevent file system corruption, it avoids repetitive unnecessary scanning by implementing partial scanning, it works transparently in kernel space, and it can be ported to many file systems.

I Voted? How the Law Increasingly Restricts Independent Security Research

Cindy Cohn, EFF

Summarized by Rachel Greenstadt

Cindy Cohn used the work Avi Rubin and his colleagues did examining the Diebold voting machines to illustrate a problem facing security researchers today: In order to do our research, we need "her or someone like her" to answer our legal questions.

The first hurdle she often faces when dealing with these matters is people questioning the need for independent security research. She often has to explain that the vendor might not otherwise be motivated to fix security problems and that, since we are all connected, good security is in the public interest. The insecure system in question in this case was the Diebold voting software. The source code was leaked on the Internet, and researchers at Johns Hopkins and Rice did an independent security audit. They found serious problems.

The talk focused on four areas of the law where the Diebold study could be called into question. These were copyright law, the DMCA, the Computer Fraud and Abuse Act, and trade-secret law. In the case of copyright law, Diebold owned the copyright to the code. In order to study the code, the researchers needed to make copies, facing as much as $150,000 in fines for infringement. In this case, the researchers had a good case for making fair use of the code, since it was for critical and scientific work and didn't impede the market. However, the law is structured poorly for protecting fair use, as there is no cut-and-dried way to tell whether an action qualifies as fair use, so you need to hire a lawyer.

Next came the DMCA. The DMCA prohibits people from circumventing a technological measure that controls access to a copyrighted

work. The scientific exception is effectively useless. Some of the files were encrypted zip archives, so the EFF instructed the researchers only to audit code that had been unzipped by others. Members of the audience expressed incredulity that a zipped archive could be considered an effective control mechanism. Another issue raised by the DMCA is publishing findings. It is unclear how far this goes: DeCSS? source code showing how to bypass some DRM? a high-level description of such code? an academic paper? This last case was at issue in the SDMI challenge paper presented at USENIX '01 Annual Technical Conference. The RIAA made threats but then backed down, so no legal precedent has been set.

The Computer Fraud and Abuse Act is the generic federal computer intrusion law. It states that unauthorized access to a protected computer is illegal. Even though the code was publicly available on Diebold's servers, someone probably broke this law to access it. There was a concern that Diebold might claim an intruder stole the code and handed it to the researchers. That would be difficult since the code was so widely available on the Internet. But the researchers had to depend on code falling out of the sky.

The last issue that often comes up is trade-secret law, which basically states that a company's desire to keep any particular information secret has to be respected and that anyone who publishes that information is liable. Many vendors would love to claim that the security flaws in their products are trade secrets. Fortunately, reverse engineering is allowable. However, not all security flaws can be found through reverse engineering, and EULA licenses often prevent reverse engineering. This institutionalizes security through obscurity.

The researchers in the Diebold situation finally got enough clearance to do their research, but there were a lot of legal pitfalls, and EFF needs support. Cindy urged the audience to support HR 107, which provides a fix to the DMCA, and said that it is important to talk about the need for security research with people who are nontechnical. A good way to frame the issue is in terms of consumer protection: people need to be protected from companies that will sell you snake oil security and buggy software. Avi Rubin mentioned that everything turned out okay in their case, but would she advise other researchers to take the same risk? Cindy responded that, at the end of the day, she does not tell people what to do. It's her job to lay out the risks, costs, and benefits and whether EFF will represent them. Great advances in the law have happened because people were willing to take chances and bear it if they lost. The history of the law demonstrates that people are rarely given rights, that they must take them. She can tell you what the law says now, but then you have to make your decision.

PROTECTING SOFTWARE II

Summarized by Stefan Kelm

Side Effects Are Not Sufficient to Authenticate Software

Umesh Shankar, Monica Chew, and J.D. Tygar, University of California, Berkeley

This talk was somewhat different from most of the other ones, in that it didn't present a solution to a security problem but presented an attack to a security solution that had been presented at the previous security symposium.

Umesh presented a number of problems with Genuinity, a system allegedly able to authenticate remote systems without using trusted hardware. Since the Genuinity source code was not available to the authors, they based their attack scenarios on what had been published already. The basic idea behind Genuinity is the computation of checksums by remote systems, which subsequently gets verified by the authenticating system. The authors described and implemented what they called a substitution attack: They computed the correct checksum quickly while running non-approved (arbitrary) kernel code. Thereafter, the original data in the computation needed to be substituted.

Umesh even went to so far as to argue that remotely authenticating systems will not be possible at all without some form of trusted hardware. Moreover, he questioned the availability of useful applications except, e.g., set-top game boxes.

For more information, contact ushankar@cs.berkeley.edu or see http://www.cs.berkeley.edu/~ushankar/research.

On Gray-Box Program Tracking for Anomaly Detection

Debin Gao, Michael K. Reiter, and Dawn Song, Carnegie Mellon University

In this more theoretical talk, Debin Gao presented work on anomaly detection systems. Today there are a number of systems trying to detect anomalies by closely monitoring system calls; these systems usually can be grouped into white box, black box, and gray box systems. The motivation behind this work was to perform a systematic study on existing systems.

During the analysis Debin and his team discovered that all the current systems can be organized along three axes: (1) the information as extracted from the process on each system call; (2) the granularity of atomic units; and (3) the size of the so-called "sliding window." In doing so they try to focus on all of the 18 resulting areas instead of looking at only one, pointing out advantages and disadvantages of one area over another. Detailed results are described in the proceedings.

In the Q&A session two attendees remarked that this work does not

take system-call parameters into account, which the author confirmed. For more information, contact dgao@ece.cmu.edu.

Finding User/Kernel Pointer Bugs with Type Inference

Rob Johnson, David Wagner, University of California, Berkeley

The final presentation of this session focused on the issue of pointers that are passed from user applications into kernel space and that might cause security problems. Misusing this "feature," an attacker might be able to read or write to kernel memory, or simply crash the machine. According to Rob, this has been and still is a persistent problem in the Linux kernel (as well as other kernels).

The solution presented by the authors uses type qualifiers that enhance the standard C language types by defining both a kernel and a user type qualifier. This has been implemented using CQUAL, a type-qualifier inference tool.

In an experimental setup, they tested their enhancements against several versions of the Linux kernel, using CQUAL in verification as well as bug-finding mode. Of the many interesting results, they presented 17 different security vulnerabilities, some of which had escaped detection both by other auditing tools and by manual audits. Moreover, all but one of those bugs was confirmed to be exploitable by an attacker. As an additional result, not in their paper, they reported that their analysis of Linux 2.6.7-rc3 revealed that although some bugs had been fixed, they were able to find seven new security vulnerabilities in only two days.

Someone asked why programmers are not using the kind of solution presented here even though similar work has been available for quite some time. Rob replied that he thinks this is beginning to change.

For more information, see http://www.cs.berkeley.edu/~rtjohnso/.

Metrics, Economics, and Shared Risk at the National Scale

Dan Geer, Verdasys, Inc.

Summarized by Serge Egelman

Dan Geer's talk centered on risk assessment as applied to national security. He covered the current state of risk assessment, the nature of risk, how various models are used to measure risk, and predictions for risk in the near future.

Geer first explained how risk assessment is related to engineering; getting the problem statement correct is extremely critical. The correct problem cannot be solved unless the right questions are asked and the problem is thoroughly understood. In terms of national security, the questions are centered around what can attack our infrastructure, what can be done about it, what the time frame is, and who will care about it.

Because of the Internet and other enabling technologies, our society has greatly increased the amount of information that is available. While this increases productivity, it also increases risk since, in an interdependent society like ours, it is now increasingly easy to stumble into different locales. The technologies are still following Moore's Law, yet the skills needed to properly use this new power are unable to keep pace. This divergence is creating a dangerous situation with broad implications.

Geer mentioned sitting in on a risk management discussion at a major bank. The key to managing their risk is knowing exactly how much risk to take. Not taking enough risk does not take full advantage of potential returns (thus losing money), whereas taking too much risk can result in insolvency. The biggest problem for them is finding this median. In finding the right amount of risk, scenarios are discussed. These scenarios cover every conceivable factor, as risk is the result of many interdependencies. This system is the basis for our banks and financial markets, and it largely works, for one reason: There is zero ambiguity about who is responsible for taking the risks. And this is a problem for national security because it is not very clear who owns the risk.

Along this line of reasoning, online risk is difficult to calculate because the Internet is a new and unique domain. The economic models function because the institutions are fairly static and predictable, whereas the Internet crosses many boundaries and jurisdictions. John Perry Barlow suggests that the Internet is separate from other jurisdictions, while others argue that it belongs to every jurisdiction. Attacks on the Internet are easy because the HTTP transport model allows the attacker to concentrate on the attack itself, as he or she does not need to worry about propagating it. Software companies are increasingly relying on HTTP for communication with applications, thus allowing attacks to be more concentrated. Additionally, this will increase the total amount of traffic on HTTP, which will make content filtering infeasible.

Economics have dictated a shift toward data. Corporate IT spending on data storage has been increasing every year, from 4% in 1999 to 17% in 2003. The total volume of data has been doubling every 30 months. Because of this, the focus of security will shift toward protecting the data. Viruses, spam, and online theft (whether it be identity or economic) have greatly increased. Users respond with complaints, and legislators respond with an increase in laws governing online security and privacy. But much of the response is ill-conceived; HIPAA attempted to account for technology change, but resulted in a law that was totally unreadable.

On the national scale, there are many considerations for mitigating risk. We create unique assets to decrease ambiguity; for example,

there is only one DNS system, since having multiple ones will defeat the purpose. But this creates a huge risk in that it offers a single target. The best counter to this is replication, creating multiple homogeneous nodes that all work together. On the other hand, replication can result in cascade failure; nodes can be used to propagate attacks. Creating this monoculture has a direct effect on exploits, since only one is needed to take out entire networks of machines. Additionally, the amateur attackers create smokescreens for the professional attackers.

Currently, the best solution we have for online attacks is field repairs. Software is patched when a vulnerability is found. This has many implications, the first of which is liability. Software is sold "as is," and thus the user agrees not to hold the manufacturer liable for any problems if the product updates are not applied. Automatic updating of software has been mandated in many license agreements for this same reason. At the same time, applying patches is not without risk. As Fred Brooks described, fixing flaws in old software invariably introduces new ones.

Geer predicts that within the next 10 years, advances in traffic analysis will greatly increase, just as advances in cryptography have increased over the last 10 years. Additionally, increasing threats will result in decreasing security perimeters. Currently firewalls are used to secure a corporate network; this will shift toward mechanisms to protect individual data. Security and privacy are currently being treated similarly; this will also change in the future.

Geer discussed further challenges in the next 10 years. Among those were eliminating large-scale epidemics and minimizing the amount of skill needed to be safe. He proposes creating commercially available tools for creating certifiable systems. Finally, he believes that information security needs to be as good as or better than financial security. But to accomplish this, better metrics are needed to answer specific assessment questions.

Because of the immediacy of this need, there is no time to create a system from scratch. We must steal models from other fields such as public health, insurance, financial management, and even physics. These models must take into account information sharing and must place values on the information. This amounts to calculating both replacement and future values. Currently the basis for these are the black market values: email address, chat screen names, and pirated software are all sold. In lieu of measuring security, we can simply assign risk in much the same way that credit card companies do (i.e., $50 liability on fraudulent purchases). These metrics are not free, however. There is always the tendency for government or other entities to impose strict controls.

In conclusion, Geer stressed that, rather than using words to describe problems, we need to do quantitative analysis. Unknown vulnerabilities are secret weapons of the attackers. The absence of a major incident says nothing about a decrease in risk. All of this is further ensuring that tradeoffs will be made between safety and freedom.

THE HUMAN INTERFACE

Summarized by Nick Smith

Graphical Dictionaries and the Memorable Space of Graphical Passwords

Julie Thorpe and Paul van Oorschot, Carleton University

Graphical passwords can be a more secure authentication mechanism than traditional text passwords, because they offer a wider set and could possibly effect a positive change on the percentage of the password set that a typical user will choose. It may also be easier for users to memorize an image than a text password.

The idea proposed is to have a user draw an image on a grid (larger or smaller to increase or decrease size of the password set) and to track the start and end position of each pen stroke to create an image definition. In this way, identical images could possibly be drawn in several different ways, which further expands the password set. The idea does suffer from the same problem as text passwords, though, which is that users are more likely to draw certain patterns (e.g., symmetric images along center axes, common images, low number of pen strokes, similar drawing sequences).

Regardless of drawbacks, which are unavoidable in any solution, graphical passwords are shown by Thorpe and Oorschot to be potentially more useful and secure than text passwords.

On User Choice in Graphical Password Schemes

Darren Davis and Fabian Monrose, Johns Hopkins University; Michael K. Reiter, Carnegie Mellon University

Recognition of images, specifically, of human faces, can be used to replace text passwords for authentication. Humans are especially good at pinpointing minute and detailed characteristics in the human face, especially in those of a person's native culture.

There are two proposed schemes: the "Face" scheme, in which a user chooses a variable number of faces from a single provided set; the "Story" scheme, where a user chooses one picture (not all of which are faces) from each of several provided sets (recorded in sequence). In a small, 154-student study done within the university, it was found that the "Story" scheme proved much more secure (i.e., more difficult to guess) than the "Face" scheme.

One drawback to these schemes is that certain races/cultures/genders

tend to choose the same type of pictures in a set (supermodels, cute animals, etc.). Also, although the "Story" scheme proved superior to the "Face" scheme, some students found it hard to remember the sequence of pictures even though they could identify all of the correct images.

Design of the EROS Trusted Window System

Jonathan S. Shapiro, John Vanderburgh, and Eric Northup, Johns Hopkins University; David Chizmadia, Promia, Inc.

The goal of this system is to impose MLS information flow controls into the X Window System. The idea is that each window's information should not be available to all other windows, but only to trusted windows (such as those within the same process). For example, common features such as cut&paste and drag&drop currently use bidirectional communication between windows, which violates the security measures of the system. Workarounds were developed to make the communication unidirectional. Also, the storage buffers used for each window must now be managed by the client rather than the window server (in this case, X). Future work anticipates the porting of current GUI toolkits.

Exploiting Software

Gary McGraw, Cigital

Summarized by Chad Mano

Gary McGraw presented an eye-opening view into the world of software security. Traditionally, security is thought of as an infrastructure problem, not a software problem. Also, many believe that cryptography is the key to providing security. However, as McGraw puts it, there is no such thing as "magic crypto dust."

The security issue that needs to be addressed concerns the software, not the infrastructure. McGraw described the "Trinity of Trouble": connectivity, complexity, and extensibility, which are the underlying reasons why developing secure software is a difficult task.

- Connectivity: The Internet is everywhere, and most software is on it.
- Complexity: As the size of the code increases, so do the number of vulnerabilities.
- Extensibility: Systems evolve in unexpected ways.

Two things that make software vulnerable to attacks: bugs and flaws. Bugs are the simple, easy-to-find mistakes in the code. Flaws are more complex problems that can be the result of mistakes in code and design. We traditionally work on fixing bugs, whereas attackers don't care whether it's a bug or a flaw that allows them to exploit software.

Educating new software developers is one of the keys to improving security. Traditional software engineering classes are boring, but that does not have to be the case. Students should be taught how to break software the way hackers do so they will better understand how to build secure software. Breaking stuff is fun and makes for a more invigorating class, as well as providing a valuable learning experience.

Functional testing and security testing are not the same thing. Functional testing looks at a program as a black box and tries to determine whether the output is correct. Security testing needs to get inside the code. Hackers use tools such as decompilers and disassemblers to understand the control flow of a program. Security testing cannot be used to build security into a program. Security must be built into the design of the program, so that security is truly inside the code.

See McGraw's book, co-authored with Greg Hoglund, *Exploiting Software: How to Break Code*, for more information on this topic.

PANEL: PATCH MANAGEMENT

Crispin Cowan, Immunix; Marcus Ranum, Bellwether Farm; Eric Schultz; Abe Singer, SDSC; Tina Bird, InfoExpress

Summarized by Eric Cronin

Crispin Cowan began the presentations by looking at the tradeoff between safety and stability inherent in patching. Patching too late leads to getting hacked; patching too soon can lead to problems from buggy patches. For a given vulnerability these two trends move in opposite directions as time passes (bugs decrease, compromises increase), so there should be some optimal point, where these two curves cross, at which to apply the patch. Unfortunately, from studying historical information, Cowan found that on average it takes 10 days after being released for the final stable revision of a patch to arrive—up to 30 days in some cases. Attacks, on the other hand, are beginning within hours of a vulnerability being published. In reality, the two curves never intersect, so from a security standpoint patch management is a doomed cause. Cowan instead suggested that patching be looked at as a maintenance activity and alternative strategies be used for security. The first alternative proposed was to only run perfect software, which has some problems in practice. The second alternative was to rely on intrusion detection and intrusion prevention technologies, such as the fine products sold by Immunix.

Tina Bird related some of her experiences as system administrator at Stanford during the MS Blaster attack. On July 16 the vulnerability exploited by Blaster was announced. On July 18, proof-of-concept attack code became available. A month later, on August 18, Blaster hit the Internet. Despite this month-long window during which users could have patched, 8,100 Windows machines at Stanford were compromised. Many users had not patched

because they did not want to break working systems. The CVE (Common Vulnerabilities and Exposures) reports announcing patches make it very difficult for users to gauge the criticality of a patch. From her experience, Bird thought the best solution in an environment like Stanford's was to remove free will from the users and use automated update systems to force patching. Bird also noted that by their nature, the most dangerous vulnerabilities are those against network services that are remotely accessible, that are on by default, and that do not require authentication. These same properties also make it very easy for administrators to scan their networks periodically and locate unpatched hosts.

Eric Schultz provided some insight on the view of patch management from "the other side." While at Microsoft he had a part in every service pack and security patch released. One strategy used by Microsoft was to remove all references from service pack release notes for fixes to previously undocumented/unannounced security fixes, in order to lessen the risks to those not applying the service pack. Microsoft usually saw a flurry of activity immediately after a patch was announced, and then little activity until right after a worm hit, at which time there was a spike in purchases of patch management solutions.

Abe Singer briefly talked about a major security attack suffered by SDSC last spring. Of note, none of the machines involved in the attack was running Windows; the vulnerabilities were all against *NIX hosts. A contributing factor to the vulnerability was that Sun's patch system indicated that kernel patches had been applied but the machines had not been rebooted to use the new kernel. Without accurate reporting it is very difficult to determine what the vulnerability status of hosts actually is.

The final presenter was Marcus Ranum, who concurred with Cowan's observation that patch management was the wrong strategy for security. He went one step further, however, and insisted that patching was a fundamentally stupid thing to do. Production Systems 101, he stated, is "If it's working, don't f— with it." He never patches his own systems, and he never changes his passwords (several of his passwords were then provided to the audience). Instead, he solved his security problem by configuring his network so that he doesn't have to care about new vulnerabilities. Corporations want the conflicting features of cheap and good software as well as unsegregated Internet-facing hosts and security. His solution is to run software that is less questionable than the rest of the herd and to run even that software within sandboxes and with extra protection. He concluded by stating that securing systems is engineering, while patching systems is anti-engineering.

In the Q&A, Steve Bellovin asked Ranum whether it was reasonable to expect every user to write her own "better than the herd" Web browser, for example. The panel answered that when you need to run commodity software such as a browser, you shouldn't run it on a machine that you care anything about; the techniques the panel had presented were suitable for servers. Another audience member pointed out that by properly isolating servers with chroot and systrace it is possible to run after a vulnerability without patching and still be secure. Another point raised was that most machines on a network are desktops and not servers, and that these machines are rarely professionally managed. When these machines are attacked, the collateral damage can impact the network and even the most hardened server. The panel agreed with this observation, and also noted that it is often wise to treat these unmanaged desktops (and laptops) as being just as dangerous as remote hosts on the Internet, and placing them outside the innermost firewall surrounding critical servers.

Military Strategy in Cyberspace

Stuart Staniford, Nevis Networks

Summarized by Marc Dougherty

Stuart Staniford presented his predictions for a hypothetical cyberwarfare scenario. There has not yet been a real cyberwar, but Staniford believes the tactics and technologies necessary will develop rapidly in the wake of real cyberwar. Staniford describes a scenario in which China invades Taiwan. The US dispatches a few carrier groups to assist the Taiwanese, causing substantial damage to the Chinese invasion force. In order to eliminate US involvement, China opens the North American theater with a cyberattack.

Staniford theorizes that in order to have sufficient results, the Chinese must disable two critical infrastructures. Staniford believes the number of targets necessary to disable an infrastructure is on the order of one hundred. The attacker not only must compromise the security of internal corporate networks, but must then gain access to the SCADA control systems in order to disable the target. Staniford estimates that a single attacking battalion could range from 150 to 1000 attackers.

Only a nation-state could muster the extensive resources necessary to create such a tremendous force. The attacking force is likely to be extremely disciplined and well trained. Because of this, Staniford believes that defensive forces must be organized and trained like a military regime, especially in the case of critical infrastructures.

Summarized by Marc Dougherty

Copilot — A Coprocessor-Based Kernel Runtime Integrity Monitor

Nick L. Petroni, Jr., Timothy Fraser, Jesus Molina, and William A. Arbaugh, University of Maryland

Nick Petroni described a Kernel Integrity Monitor known as Copilot, which resides on a PCI card. The goals of Copilot are to create a system that is compatible with commodity hardware, effective in detection as well as performance, and isolated from the host, and that cannot be tampered with in the event of a compromise. The current system functions on commodity x86 Linux systems without modification.

Copilot monitors a number of data points, including the list of loaded kernel modules and a hash of the Linux kernel text. The results of these checks are compared to a known good state, and in the event of a change, it is reported to the administrative system, to which the Copilot PCI card connects directly.

In an experimental test of Copilot's capabilities, it was able to detect all 13 of the rootkits attempted. The performance penalties of using Copilot are also relatively minor. When checking every 30 seconds, there is only a 1% performance penalty. Continuous checking, however, showed a 14% penalty, which is believed to be caused by bus contention.

Further work for the Copilot team includes predicting memory footprint based on a binary, and instituting further checks on kernel data such as the process table and open file descriptors. Copilot can also currently be extended to support dynamic data.

Fixing Races for Fun and Profit: How to Use access(2)

Drew Dean, SRI International; Alan J. Hu, University of British Columbia

Drew Dean presented a probabilistic solution to the age-old problem of race conditions. Dean's solution deals particularly with the problem of setuid programs using the access rights of the real user. The `access()` system call was created for this purpose, but due to the non-atomic nature of the call, it is still vulnerable to a TOCTOU attack (time-of-check to time-of-use).

Dean's approach involves applying "hardness amplification," a technique commonly used in the cryptography realm, to make winning this type of race more difficult. The system described uses a coefficient K to determine the strength of the system. When opening a file, `access()` is called, and the file is opened. Instead of using that file descriptor, the following steps are performed K times.

- Random delay
- Call `access()`
- Random delay
- Re-open the file
- Verify that the device, inode, and `fstat()` results match
- Close the re-opened file

This process relies on `access()` producing no side effects, `open()` being idempotent, and `fstat()` not being vulnerable to a race condition. The result of this procedure is that an attacker must win 2K + 1 races in order to achieve his goal. Dean recommends a K of 7 for single-processor machines, and a K of 10 for a reasonable level of security.

Network-in-a-Box: How to Set Up a Secure Wireless Network in Under a Minute

Dirk Balfanz, Glenn Durfee, Rebecca E. Grinter, Diana K. Smetters, and Paul Stewart, PARC

Diana Smetters presented a PKI-based wireless security solution modeled on the 802.1x EAP. The barrier preventing widespread use of EAP is the configuration, which involves typing large hexadecimal strings into the clients and the APs.

The goal of NiaB is to create a smarter access point that handles the generation of certificates and exchange this new authentication information through a second communication channel. By using a short-range method such as IR for this second channel, wireless security becomes a matter of physical security.

The NiaB architecture is divided into four parts:

- Wireless Access Point
- Enrollment Station
- Certificate Authority
- Authentication Server

When joining a network, the user interacts with the Enrollment Station portion of the system, which talks to the CA to autocreate client keys. Once the keys have been exchanged, the user can safely communicate with the Authentication Server element over TLS, since the public key digest is also exchanged via the secondary communication channel. This communication is done using a custom protocol called EAP-PTLS.

Smetters also noted that for enterprise use, the architecture can be separated to allow a centralized Enrollment Station, CA, and Authentication Server. Experimental results show that users can connect to a NiaB network in under one minute, demonstrating that wireless networking can be usable as well as secure.

Design and Implementation of a TCG-Based Integrity Measurement Architecture

Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn, IBM T.J. Watson Research Center

Reiner Sailer presented an integrity detection system based on the work of the Trusted Computing Group. The TCG system relies on a special chip that stores information about the boot process. The system presented by Sailer uses this same TPM chip to detect tampering with

important system files. TPM chips are already installed in most IBM laptops.

This system uses the TPM chip to store SHA1 checksums of important system files. These checksums are then digitally signed by the chip. The checksums are also available to the kernel of the host OS.

The software portion of the current implementation consists of a kernel module which performs the above checksumming before opening each file. Experiential results show that a clean hit takes only .1 millisecond, and the opening of a new file takes only 5 milliseconds plus the time required to perform the SHA1 checksum.

This work is included in the TCG Linux project. More information is available at http://www.research.ibm.com/secure_systems_department/projects/tcglinux/.

What Biology Can (and Can't) Teach Us About Security

David Evans, University of Virginia

Summarized by Alvin AuYoung

Note: Slides from the talk are available at http://www.cs.virginia.edu/~evans/usenix04.

Nature provides a fascinating example of how to design computer systems. David Evans presented analogies between nature and computer systems, highlighting both the potential and the limitations of using nature to guide the design of secure computer systems.

Approximately 99.9% of species in nature become extinct. Evans emphasized the slim odds of survival faced by nature's species, who must adapt in order to survive their predators and other attacks in nature. He drew a parallel between attacks faced by these species and well-known attacks faced by today's computer systems. For example, a lion chasing down a gazelle might be similar to a "brute force attack," where an attack is conducted by sheer force. A Bolas spider attracting male moths by emitting chemicals that mimic pheromones of female moths is similar to a communication integrity attack.

Species that are able to survive against predators in nature and various ecological phenomena are examples of secure and robust biological systems. Understanding how these species have evolved can help us design secure and robust computer systems.

Biological systems are too complex to understand analytically. Instead, Evans suggests that we should learn from the qualities of existing species in nature that have survived over time. There are four common traits observed in such systems: expressiveness, redundancy, awareness of surroundings, and localized organization. If we were to mimic these characteristics, could we also build programs that survive both its adversaries and the test of time?

Amorphous computing, swarm programming, and autonomic computing are examples of research that attempts to use these characteristics to design computer systems. In particular, Evans discussed the amorphous computing project from the lab of Radhika Nagpal, where the goal is to design a system in which a potentially large number of simple and possibly unreliable pieces can cooperate to create a system with complex and well defined behavior. An example of this is Nagpal's work with origami mathematics. In this work, a set of identically programmed cells forms a sheet of (virtual) paper, and the goal is to have these cells cooperate using only local interactions to form a complex structure such as an origami shape (e.g., half of the cells will "move" to one side to simulate a fold in the paper). Although this is an example of how to design a robust system that captures some of the qualities of nature's systems, there is still a long way to go before we understand how to use these qualities to design secure systems for real-world applications.

Another place where the principles of nature's systems can be applied to computer systems is computer immunology. Identifying intrusion and removing viruses in computer systems is similar to how our immune system wards off attacks. In nature, diversity of receptors within species allows them to survive strains of specific pathogens. As a result, a large number of the members of a species can survive. In computer systems there has been a push toward providing system diversity. A commonly cited example is in operating systems, where over 90% of the machines in the world use Microsoft Windows. Largely as a result of this, computer worms, using a single vulnerability in Windows, have been able to infect a large number of computer systems connected to the Internet. The fact that so many machines on the Internet share vulnerabilities has allowed Internet worms to proliferate and has caused many people to push for diversifying the operating systems market. Evans argues that diversifying an operating system isn't the complete answer. For example, vulnerabilities in widespread network protocols like TCP/IP, applications like the BlackICE firewall, and common libraries like libpng also contribute to this problem. The lesson here is that diversity must be present at several levels of abstraction in order to be effective. Evans cites existing work such as instruction-set randomization and protection of special registers in an address space as examples of diversity techniques in computer systems.

There are limits to the lessons to learn from nature. In particular, Evans notes that computer systems are human-engineered and therefore don't naturally "evolve." In other words, the process of evolution is smarter than we (humans) can be in redesigning our systems, so we shouldn't count on a few iterations of software to evolve into secure software. There are also

many instances where nature fails, such as mass extinction of species due to environmental changes and the ability of viral epidemics to wipe out large portions of a species' population. Evans also notes the success of humans in engineering mass destruction in nature, such as vaccines to wipe out smallpox. He uses these points to establish that nature can succumb to "out-of-band" attacks (environmental changes, humans).

In conclusion, Evans says that while there is still much we can learn from biological systems, we have to do much better than nature in designing computer systems, because the attacks that our systems face can also include "out-of-band" attacks that nature often cannot defeat.

FORENSICS AND RESPONSE

Summarized by Zhenkai Liang

Privacy-Preserving Sharing and Correlation of Security Alerts

Patrick Lincoln, Phillip Porras, and Vitaly Shmatikov, SRI

The prevalence of computer viruses and worms drives the need for Internet-scale threat-analysis centers. These centers are data repositories to which pools of volunteer networks contribute security alerts, such as firewall logs or intrusion alerts. However, the possibility of leaking private information in the contributed data prevents people from providing alert-sharing data for those centers. Vitaly Shmatikov presented a practical scheme to provide strong privacy guarantees to those who contribute alerts to these threat-analysis centers.

The authors' solution is an alert-sharing infrastructure, the core of which is a set of repositories where alerts are stored and accessed. In preventing sensitive information (e.g., IP addresses, captured and infected data, system configuration information, defense coverage) from being revealed, several techniques to be used in combination are proposed, including scrubbing sensitive fields, hashing IP address, re-keying by repository, using randomized host list thresholds, and delaying alert publication. To prevent single point of failure, they use multiple alert repositories and randomized alert routing. The alert sanitization techniques protect information in raw alerts, as well as allowing a wide variety of large-scale and cross-organization analyses to be performed. In the performance evaluation, the authors conclude that the cost of their scheme is very low: "a small impact on the performance of alert producers, and virtually no impact on the performance of supported analyses." It is implemented as a Snort plug-in for alert sanitization.

Q: How do you defend flooding of a repository?

A: There's no defense against flooding currently. One possible solution is to let the contributors of alerts register with the system.

Static Disassembly of Obfuscated Binaries

Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna, University of California, Santa Barbara

Christopher Kruegel introduced a static binary analysis technique for disassembling obfuscated binary codes. Disassembly is the process of recovering a symbolic representation of a program's machine code instructions from its binary representations. Recently, a number of techniques have been proposed to make it difficult for a disassembler to extract and comprehend the program's high-level structure while preserving the program's semantics and functionality.

Without the requirement that the code was generated by a well-behaved compiler, their disassembler performs static analysis on Intel x86 binaries. The analysis techniques can be divided into two classes: general techniques and tool-specific techniques. General techniques do not rely on any knowledge of how a particular obfuscator transforms the binary. These techniques first divide the binary into functions. Then they attempt to reconstruct the functions' intra-procedural control flow graph, which is a major challenge of their approach. Finally, a gap completion step improves the result of the previous step. In their tool-specific techniques, the general techniques are modified to deal with binaries transformed with Linn and Debray's obfuscator.

The authors' techniques were evaluated on eight obfuscated SPECint 95 applications. The authors claimed that their disassembler provides a significant improvement over the best disassembler used in the evaluation by Linn and Debray. With the tool-specific techniques, they said that the binary was disassembled almost completely (97% on average). The authors also performed experiments to find the ratio between the number of valid instructions identified by the control flow graph and the number of valid instructions identified by the gap completion phase. In their observation, the control transfer instructions and the resulting control flow graph constitute the skeleton of an analyzed function.

Q: Can you comment how your tool works on encrypted or encoded binaries?

A: The tool is currently a static analysis tool, which cannot help in this situation. It can be extended with dynamic disassembling.

Autograph: Toward Automated, Distributed Worm Signature Detection

Hyang-Ah Kim, Carnegie Mellon University; Brad Karp, Intel Research and Carnegie Mellon University

Hyang-Ah Kim described Autograph, a system that automatically generates signatures for novel Internet worms that propagate

using TCP transport. The paper answers the question, "How should one obtain worm content signatures for use in content-based filtering?" The authors noted that all current content-based filtering systems use databases of worm signatures that are manually generated. On detecting a virus, network operators communicate with one another, manually study packet traces to produce a worm signature, and publish the signature to worm signature databases. This process of signature generation is slow in the face of today's fast-propagating worms.

Autograph takes all traffic crossing an edge network's DMZ as the input, and outputs a list of worm signatures. A single Autograph monitor's analysis of traffic is divided into two main stages. First, a suspicious flow selection stage uses heuristics to classify inbound TCP flows as either suspicious or non-suspicious. Then Autograph selects the most frequently occurring byte sequences across the malicious flows as signatures. To do so, it divides each suspicious flow into smaller content blocks using content-based payload partitioning (COPP) and counts the number of suspicious flows in which each content block occurs. The authors also created an extension to Autograph, tattler, for distributed gathering of suspected IP addresses using multiple monitors.

In the evaluation of local signature detection, the authors conclude that, as content-block size decreases, the likelihood that Autograph will detect commonality across suspicious flows increases, so it generates progressively more sensitive but less specific signatures. In the evaluation of distributed signature detection, the authors conclude that multiple independent Autograph monitors clearly detect worm signatures faster.

Q: Do all possible common signatures fall into a single-block content signature?

A: There is a possibility that we miss signatures contained in multiple packets.

Q: Do you look at multiple content blocks for signatures?

A: We are exploring this.

Q: How do you deal with the situation where someone is trying to overload (flood) the monitors?

A: It can happen. The solution is to use distributed monitors to verify signatures.

Nuclear Weapons, Permissive Action Links, and the History of Public Key Cryptography

Steve Bellovin, AT&T Labs—Research

Summarized by Tara Whalen

Steve Bellovin gave the audience an entertaining and informative trip through nuclear weapons technology, Cold War politics, and military history, with a side trip to visit Dr. Strangelove. He provided a thorough discussion of permissive action links (PALs), which were designed to prevent unauthorized use of nuclear weapons. Appropriately enough for a security forum, he proposed an intriguing theory that public key cryptography may have been developed initially as a necessary component of nuclear weapon security.

Historically, permissive action links were required to protect nuclear bombs, not from unauthorized use by enemies or rogue US troops, but to keep them from being misused by US allies. In the 1950s the US was concerned that the Soviet Union was going to invade Western Europe. Keeping a large number of US troops in Europe was not acceptable. The alternative was chosen to create NATO and provide nuclear weapons for NATO allies. However, because of political instability in some European countries and fear that the US could not tightly control deployment of these remote weapons, PALs were added to provide some measure of security.

Out of intellectual interest, Bellovin began independent research into how PALs worked; the documents he obtained from government agencies gave some hints but no conclusive information. The safety features of nuclear bombs were reasonably well documented. These features included a strong link/weak link pair: the strong link provided electrical isolation, while the weak link was designed to fail under stress (such as in an accident). One component of the strong link was the unique-signal generator, a safety system designed to produce a 24-bit signal that was highly unlikely to occur by accident (meant to indicate human intention to deploy). The unique-signal discriminator is designed to lock up if it receives a single erroneous bit.

The PALs themselves were originally electromechanical controls which required two cooperating parties to unlock the weapon. PALs respond to a variety of codes (recently, 6- or 12-digit codes); the weapon is designed to deactivate if too many incorrect codes are entered. Bellovin speculated on design principles that may have been used for PALs. One hypothetical design involves encrypted signal paths: PALs use switches to control the voltage path to the detonators. The original designs used rotors, similar to WWII-era encryption machines, but more recent models use a microprocessor to control the switches; possibly this works by providing an encryption key for the weapon's environmental sensors, with the signal being decrypted by the strong links. Another hypothetical design involves encrypted code, in this way: It is highly likely that the internal control and sequencing

requirements are complex; this timing information, or perhaps the code paths, is encrypted. The public interface could be used for rekeying plus decryption.

Bellovin brought the discussion back to cryptographic history, stating that he found no requirement for the use of public key crypto in PALs (or any indication it was used there, except in one prototype). The PAL sequences are short, which indicates that no secure public key cryptosystem could have been supported. However, consider that by law US nuclear weapons can only be deployed by order of the President; to support this requirement for authenticating the arming code, it is possible that the NSA invented digital signatures. (Added Bellovin wryly, "Including an X.509 naming format for bomb certificates.") In addition, codes need to be put into the PAL: How is this done securely? Since this information should not be passed in the clear, this requirement may have led to the NSA's development of public key cryptography. (This theory is not well documented, but makes for interesting speculation.)

What lessons can security professionals learn from nuclear weapons safety and security designs? One important lesson is to understand exactly what problem you're solving: The weapons designers got it right because they knew precisely what the real problem was. For access control, you want to find the One True Path that everything must pass through, and block that.

For more information on this topic, see Steve Bellovin's PAL Web page at http://www.research.att.com/~smb/nsam-160/pal.html.

DATA PRIVACY

Summarized by Rachel Greenstadt

Fairplay—A Secure Two-Party Computation System

■ ***Awarded Best Student Paper!***

Dahlia Malkhi, Noam Nisan, and Yaron Sella, Hebrew University; Benny Pinkas, HP Labs

Yaron Sella presented an implementation of Secure Function Evaluation (SFE), a theoretical construct studied in the cryptography community for 20 years. The classic example of SFE is the millionaire's problem. Two millionaires want to know who is richer. One has $X million and one has $Y million, and they don't trust each other. They need either a trusted third party or SFE. In this case the function is > = <, but it could be any function. The goal of the research was to see whether two-party SFE was practical and to better understand where the bottlenecks were, in computation or communication. They wanted to push this protocol from the theoretician's realm to the practical arena.

They defined two languages, Secure Function Definition Language (SFDL) for specifying the function to be evaluated, and Secure Hardware Definition Language (SHDL) for turning the program into a circuit. Thus if Alice and Bob want to evaluate a circuit, Bob takes the circuit and garbles it *m* times and sends these to Alice. Alice chooses one and tells Bob which. Bob then reveals the secrets of all the non-chosen ones and Alice verifies that they are okay. This is the commonly used cut-and-choose method of indirect randomized verification. Bob types in his input and sends the garbled version and Alice types her input. They engage in oblivious transfer where Bob is the sender and Alice is the chooser. Alice evaluates, sends to Bob, and they both print the results.

Yaron then gave the audience a live demo of Fairplay. He first showed examples of SFDL and SHDL (the millionaire's problem) and created circuits. He then showed examples of running the billionaire's problem (bigger numbers). There was a short delay as the circuits were sent; Bob had 100000, Alice 100001. So Bob received 0 and Alice 1. The code was written in Java, using its crypto-libraries and SHA1. They tried several functions: AND, billionaire's problem, keyed database search, and finding the median of unified arrays. The main bottlenecks were in the communication of the circuits and the oblivious transfer. You can play with the system at http://www.cs.huji.ac.il/danss/Fairplay. An audience member suggested that the delay experienced on the LAN might be reduced by turning off nagling.

Tor: The Second-Generation Onion Router

Roger Dingledine and Nick Mathewson, The Free Haven Project; Paul Syverson, Naval Research Lab

Roger Dingledine began by explaining that TOR (The Onion Routing) was similar to Zero Knowledge's Freedom networks, except with some improvements and free. It aims to provide anonymity by protecting packets from traffic analysis. The research is funded by the government (they want to look at Web sites without revealing their .gov and .mil addresses and to protect their networks from insider attack), but anonymity is also useful for individuals, corporations, and even law enforcement. People are worried about criminals, but they already have anonymity (Nick Weaver can tell you more about this). The idea behind the network is to distribute trust among several nodes and that not all of them will collude and do traffic analysis on the users. The system provides location anonymity, not data anonymity. You should use something on top of it to scrub your cookies and whatnot.

Onion Routing is an overlay network of anonymizing proxies using TLS between each pair of links. Alice, the client, connects to the first hop, negotiates a key, then tunnels to the second hop, and then to the third hop. Each node only knows its predecessor and successor. Once built, you can multiplex TCP streams over these circuits, which are rotated periodically. To connect, you contact the directory servers (more trusted nodes) and they tell you about who's in the network, keys, exit policies, etc. There's a human in the loop to prevent Sybil attacks. They have 50 servers. TOR is 26,000 lines of C code that runs in user-space. The client looks like a socks proxy, so they don't need to build an application for each protocol they want to support. There are flexible exit policies to deal with abuse and legal issues. The system works with only a couple of seconds of latency. The system also supports location-hidden services, so Bob can run an Apache without revealing his IP. At http://freehaven.net/tor there are specifications, a second implementation of the TOR client, a design paper (this one), and code.

People asked about abuse issues, and Roger answered that there haven't been any, but they are a research network and they block SMTP. Someone asked whether someone could follow the packets by comparing their sizes. Roger answered that they already give up if the adversary can see both sides of the connection; their goal is to prevent all attacks that are easier than that. Someone wondered whether you should send the hostname or the IP address through TOR. The answer is, the hostname; the resolve has to be done at the end or DNS (since it is UDP) will break anonymity.

Understanding Data Lifetime via Whole System Simulation

■ *Awarded Best Paper!*

Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum, Stanford University

Jim Chow began by laying out a common scenario: a user at a password prompt. The question was, after you type your password, what happens to this sensitive data? Under the assumption that the software was written with security in mind, it will read the password into a buffer locked in memory, use encrypted software, and erase the password once it's done with it.

If your computer is broken into at a later date, that sensitive information is assumed by most people to be gone, but this paper showed that not to be the case. Even if the application has been careful, many extra copies of the data often exist in the tty buffer, the X server, gtk+ buffers, etc. The problem with data lifetime issues is that you can only erase the copies you know about; the others may get written to disk.

To address this problem, they wrote TaintBochs, a program to do whole system simulation and track sensitive data as it flows through software. However, just figuring out what was a copy was nontrivial; where this was the case, the program erred on the side of caution.

They learned that there were usually several copies of data stored in the kernel, and although kernel memory isn't paged, there are plenty of ways to dump it to disk, from hibernating laptops to crashes. Since indirection is so popular, it's impossible to avoid middlemen processes, so you get a lot of copies there, too. Fixing things is usually easy, though: Zeroing and string destructors destroy most taints, and TaintBochs can help with that.

During Q&A, Jim pointed out that there are possibly false negatives in the experiments, since TaintBochs can only prove the existence of taints, not prove you don't have problems. Someone else brought up the point that sometimes the compiler doesn't obey the `memset` command, but there are lots of ways to get around that problem.

My Dad's Computer, Microsoft, and the Future of Internet Security

Bill Cheswick, Lumeta Corp.

Summarized by Serge Egelman

The biggest threat to Internet security is not large corporate computers that are specifically targeted by intruders, but individual home machines that are simply configured poorly. Bill Cheswick elaborated on this by using his father as the typical user. He's an 80-year-old veteran who runs Windows XP. He uses the computer to trade stocks, check email with Microsoft Outlook, and chat with family and friends via AOL Instant Messenger. He has been "skinny dipping" for years, by which Cheswick means that his father does not have a firewall or any spyware protection, and he never updates his anti-virus definitions. Pop-ups appear on the screen every three minutes or so, but his father simply dismisses them without giving any thought to it. In fact, a computer repair person came over to take a look at the machine and discovered a variety of spyware programs, redirections to obscene Web sites, and viruses that he had never heard of before.

The question is, why should he care? He found a way of getting around all of these distractions so that he could still be productive, and he did not want a system administrator changing any of his personal settings. Since malicious attacks are very rare, he did not see any real reason to bother changing any of his habits. On the other hand, his machine had become a slave that could be used in propagating spam or denial of service attacks across the Internet, thus disrupting other people's work. Cheswick described it as a "toxic

waste dump spewing blue smoke across the Internet." It is in everyone's best interest that these machines are secured.

Computer security has become an arms race. Originally anti-virus software worked by looking for specific signatures in the virus code, but this was later thwarted by viruses that would encrypt or recompile themselves. So the detectors started running simulations to see how the virus would act on the program, but now newer viruses can detect these simulations. Trusted computing offers a solution to this problem, but it is only a matter of time before the virus writers regain control. Similar arms races can be seen in password sniffing, password cracking, hardware vulnerabilities, and even software upgrades. In an homage to The Karate Kid, Cheswick quotes Mr. Miyagi: "The best block is not to be there." That is, the key is to stay out of the game altogether.

In order to keep Dad out of the game, better client security is needed. Windows ME has eight ports open by default, and Windows 2000 and XP have increasingly more. A similar machine running FreeBSD will have one open for SSH, if any. It is obvious that these ports should not be open by default. Personal firewalls try to solve this problem, but additional software really is not needed here; the problems are caused by all the programs that are not used.

Cheswick has a solution, though: Windows OK. This could be a new end-user-oriented operating system that functions as a thin client. No firewall or anti-virus software will be necessary, since no ports will be open by default. Additionally, all programs will be signed by trusted parties, and all security settings will be located in one central location. Additionally, macros will not function in Word or PowerPoint and can only be used for arithmetic in Excel.

Such a system would be immune to the vast majority of security threats on the Internet. Cheswick believes that we may never win the malware detection arms race, so instead we must win the protection game. Unfortunately, most users simply do not see the problem; a fully hosed computer may still seem functional to them. What the user does not know is that his system is affecting other people. By creating a usable system with tight security by default, we can keep the average user out of the security game altogether.

For WiPs and Posters, please see http://www.usenix.org/events/sec04/tech/wips/.

For audio files of some sessions, see http://www.usenix.org/publications/library/proceedings/sec04/tech/mp3/.