Roving Bugnet: Distributed Surveillance Threat and Mitigation

Ryan Farley and Xinyuan Wang

Abstract Advanced mobile devices such as laptops and smartphones make convenient hiding places for surveillance spyware. They commonly have a microphone and camera built-in, are increasingly network accessible, frequently within close proximity of their users, and almost always lack mechanisms designed to prevent unauthorized microphone or camera access.

In order to explore surveillance intrusion and detection methods, we present a modernized version of a microphone hijacker for Windows and Mac OS X. This attack can be executed as soon as the target connects to the Internet from anywhere in the world without requiring interaction from victimized users. As the attacker compromises additional machines they are organized into a botnet so the attacker can maintain stealthy control of the systems and launch later surveillance attacks.

We then present a mechanism to detect the threat on Windows, as well as a novel method to deceive an attacker in order to permit traceback. As a result of the detection mechanism we address a missing segment of resource control, decreasing the complexity of privacy concerns as exploitable devices become more pervasive.

1 Introduction

The capabilities of spyware have expanded as always-on Internet connections have become increasingly frequent [11, 30]. It's not only data stored on the compromised machine that is at risk. Variants of spyware that provide audio and video surveillance

Ryan Farley
George Mason University, Department of Computer Science, Fairfax, Virginia, USA, e-mail: rfarley3@gmu.edu

Xinyuan Wang
George Mason University, Department of Computer Science, Fairfax, Virginia, USA, e-mail: xwangc@gmu.edu

through peripherals such as microphones and web-cams have been around for over ten years. This may all sound like old news, but that is deceivingly wrong.

There are a few factors why well structured surveillance attacks are only a recently growing concern and an increasingly unchecked threat reaching critical potential. Primarily, consumers are realizing that a smartphone with an unlimited data plan is almost as vulnerable as a desktop on broadband at home [19]. Also laptops, which have long had built-in microphones and Internet accessibility, are recently also being sold with built-in web-cams. Protection is even more of a concern in the modern computing environment where new regulations are constantly driving up the accountability of organizations for the loss of private data.

It is important to point out that we are not implying that surveillance spyware will be as widespread as other malware. A microphone in every house with Internet access is of little use to the average attacker and surveillance attacks will probably involve specific victims known to the attacker. This does not diminish how universal of a threat this is, after all, potentially anyone is capable of gaining an unwanted stalker, jealous spouse, or generally becoming the target of espionage [20].

The most plausible use of surveillance spyware across a set of devices is to provide a roving bug. This is a term used for audio surveillance that follows a particular victim regardless of which device they are using. If the attacker has compromised a victim's home computer, work laptop, and smartphone, then the attacker would have a greater capacity to continuously monitor the victim.

To investigate feasible methods for surveillance threats, we have implemented a complete remote attack and control package called the *roving bugnet* that approximates observed distributed control systems. The bugnet consists of a scalable number of compromised devices called *bugbots* which can stream live microphone data to a remote attacker either continuously or for a set time. To modernize older surveillance programs, our prototype can automatically compromise a vulnerable Windows (95–Vista) or Mac OS X laptop and stealthily seize control of its microphone without any action by the victim as soon as the laptop connects to the Internet.

It appears that no existing malware defense provides a generic intrusion detection mechanism against the bugbot attack. To resolve this we present a preliminary mitigation mechanism that is designed to be compatible with most Windows platforms. It can detect a process that is actively using the microphone and allow the user to set access controls. This mechanism includes a novel method to deceive a remote attacker after detection by transparently replacing the microphone input with arbitrary and realistic decoy audio. The process would trick the attacker into believing that the bug is working, yet prevent any confidential information leakage and provide time to trace the connection back to its source in order to discover the attacker.

The rest of this paper contains implementation and testing scenarios as well as design challenges and considerations. In section 2, we discuss the design and implementation of the bugnet surveillance system. A demonstration of the system is presented in section 3. Then in section 4 we introduce the prototype detection and defense mechanism as well as detail the experiments performed to test its effectiveness. We discuss related work in section 5, and conclude in section 6 with the implications and open work for this low-cost high-reward threat.

```
 1 start data handling thread
 2 open UDP server for UI
 3 fill in WinAPI structures
 4 WinAPI waveInOpen
 5 WinAPI waveInAddBuffer
 6 WinAPI waveInStart
 7 while listen for control input
 8   if input is stop recording
 9     WinAPI waveInStop
10     cleanup and exit
```

(a) Control thread for main loop and receiving user control input.

```
1 if using file open output file
2 if using network create socket
3 while WinAPI GetMessage
4   if MM_WIM_DATA
5     if using network
6       send data to destination
7     if using file
8       write data to file
9     WinAPI waveInAddBuffer
```

(b) Data handling thread for receiving messages from the sound card driver.

Fig. 1 Code overview of each waveIn recording application thread.

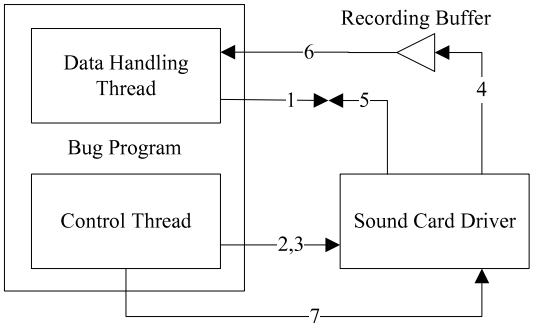


Fig. 2 Visualization of the bug program control flow. 1) Data handling thread waits for messages. 2) Control thread sets recording parameters and 3) initiates recording. 4) Sound card writes to buffers and 5) sends a message when each is full so that 6) the bug can access the recorded data. 7) Control thread stops recording when finished.

2 Roving Bugnet Design

In order to remotely monitor a target with a distributed surveillance system there needs to be two functional components: one that accomplishes a microphone hijacking, and another that maintains stealthy remote control of a compromised system. Both the outdated trojans BackOrifice and SubSeven provide remote access and microphone recording plugins, but cannot stream live data. A modern trojan, Poison Ivy includes live streaming, but requires the victim to be actively connected in order to start and stop the microphone recording. Also, since these are all trojans, by definition they are designed to require victim interaction to infect a vulnerable host.

In this section we will introduce an updated remote surveillance package that can infect Internet connected hosts without victim interaction and provide persistent management access. The attacker can instruct hosts to turn on the bug at an arbitrary time or at particular system conditions and record for either an indefinite or specified duration. This roving bugnet design is presented in two layers: the prototype microphone surveillance program in section 2.1; and, the remote management botnet in section 2.2. The layered approach of this design allows a single generic cross-platform bot to employ OS specific microphone recording executables. In fact, while this paper only details Windows XP, we have already tested the versatility of this design by successfully implementing bugbots on Mac OS X as well.

2.1 Bugbot: Microphone Access

To develop the bugbot program we used the Microsoft Platform SDK, a free Windows application programming interface (WinAPI) that can be used for multimedia application development. Of the WinAPI functions, the waveIn set was selected since it has greater flexibility and the widest compatibility for Windows versions (from 95 to Vista) and existing hardware.

Figures 1 and 2 illustrate how the program is divided into two threads. It is worth noting that Mac OS X microphone recorders use this two thread approach as well. The primary thread, figure 1(a), acts as a control and the secondary, figure 1(b), handles the data returned from the sound card.

In figure 1(a) the control input handled at line 7 is from an UDP server. This allows interactive remote control such as starting and stopping a recording. It also controls switching between using a network socket for a live data feed, writing to a file for later retrieval, or both. Line 3 sets a data structure which contains the recording parameters for the raw data that will be returned from the sound card and is set by the call in line 4. Line 5 allows for a continuous stream of data by initializing a cyclical set of buffers. Finally, the sound card is instructed to begin and end the recording through the functions in lines 6 and 9 respectively.

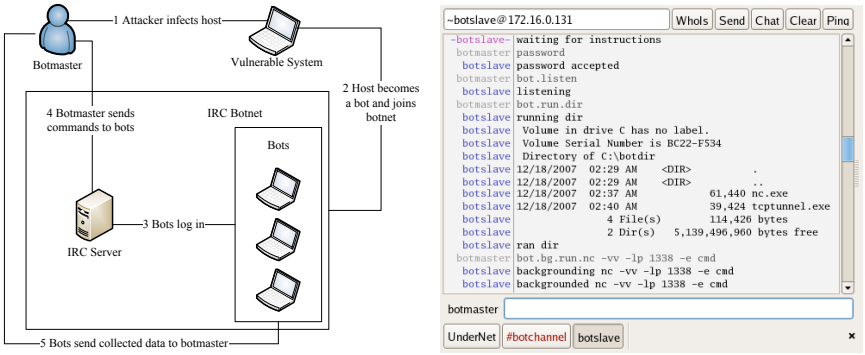
When a buffer is filled by the sound card driver, a `MM_WIM_DATA` message is sent to the bugbot process. The data handling thread, as seen in figure 1(b) loops at line 3 on a blocking function which waits for messages. `MM_WIM_DATA` messages contain the recently filled buffer's location in memory and the size of the data stored in the buffer, allowing the process to access and output the data. Line 9 replenishes the cyclical buffers initially set by line 5 of the control thread.

As an added advantage the program can detect if the network connection dies and act appropriately. If the system call to send socket data fails, then a network accessibility test is run. If that test fails, then the application will output to a file until the network connection is restored. The attacker would wait for the machine to return onto the network, and then transfer the file for local playback.

This is just one piece of the overall puzzle. While the bugbot program is fully functional for microphone surveillance, it lacks a way to install itself on the victim's host, start itself to begin with, and easily manage multiple nodes. To accomplish a complete distributed surveillance system there needs to be a remote access framework, such as the method described in the next section.

2.2 Bugnet: Remote Control

An Internet Relay Chat (IRC) bot is a program or collection of scripts that acts on behalf of an user client. The goals of IRC bots vary widely, such as automatically kicking other users off or more nefarious things like spamming other IRC users. In this paper, a free standing IRC bot is presented that monitors an IRC channel for commands from a particular user and responds accordingly.



(a) Overview of an IRC Botnet.

(b) An example of the botmaster interacting with a bot.

Fig. 3 Botnet overview and sample control session.

A botnet is a collection of bots, usually under the control of a botherder, or botmaster, using a communication method, such as IRC, to execute actions in proxy on the bots [23]. The overall structure resembles figure 3(a). Plausible purposes of botnets are click-fraud, DoS attacks, and distributed processing. The general motivation of the botmaster is to acquire as many machines as desired and maintain control for either resale or some ulterior purpose [15].

While there are many preexisting IRC bots freely available online that could be adapted for this threat, for simplicity and greater control we developed our own from the ground up. The Windows version is written in C, and the OS X version is written in Perl to support both PPC and Intel platforms.

Our IRC bot has a limited set of procedures relating to controlling who can give the bot commands, obtaining the bot's status, and running arbitrary commands on the infected host at specified times. Only once a password and the botmaster's username are approved can the botmaster issue commands. For additional functionality, the IRC bot accepts any file transfers from the botmaster username using the Direct Client to Client (DCC) protocol and stores them into the working directory for later access. To facilitate self installation, when first executed the bot copies its executable into a hidden directory and establishes itself as a service to be started on each boot-up.

The following subset of the commands we have implemented on the bugbot represents a suggested minimum for bot development: `<password>`, authenticates nick as the botmaster if the password is correct; `bot.listen`, start to accept commands; `bot.stats`, report system status and details; `bot.die`, kill self; `bot.[un]install`, run the install or uninstall routine manually; and, `bot.[bg.]run.[at<time>.]`, execute an arbitrary command, optionally in the background or at specified time.

Deciding on an infection vector to get the bot onto the target machine would need to vary by specific target; it should be noted however, that with a properly configured rootkit, the bot should remain undiscoverable on the victim's system [38].

3 Threat Demonstration

One goal of this paper is to present a viable example of a roving bugnet by means of a prototype demonstration. In this section we show each step of the entire life cycle of an example attack that can be adapted for other platforms. First, in section 3.1 we describe a method to remotely infect a Windows PC with an IRC bot. Second, in section 3.2 we show how the bot gains control of the microphone by installing the recording program and becomes the bugbot.

3.1 Infection Vector

It is possible for the attacker to use a variety of methods to get the spyware onto a victim's machine. Since advanced infection methods are beyond the scope of this paper we selected Metasploit's command line interface and the upload and execute shellcode as the payload. In order to use a familiar exploit, a default installation of Windows XP SP1 is exploited using the MS06_040 vulnerability module. All an attacker needs to do at this point is specify the bot executable as the local file that will be uploaded to the target and executed on it.

Once the bugbot is installed, it will attempt to join the botnet. At this point an IRC server is needed where the bot is programmed to look. The bot will then log in, join the predetermined channel, and post a message showing that it is ready to accept commands from the botmaster and that it can control the microphone.

3.2 Controlling the Microphone

After the bot has joined the IRC channel, the botmaster can interact with it using the commands listed in section 2.2. A basic session would resemble figure 3(b). When the attacker wishes to gain microphone control, the bug executable needs to be transfered to the compromised machine. For this implementation the attacker transfers the file to the victim using IRC DCC. Alternatively, it could be included in the original uploaded installation routine or downloaded with TFTP or FTP, both of which are included in default installs of Windows XP and Mac OS X.

With this level of remote control on each node within the bugnet, the attacker can now easily execute the surveillance program and activate the bug on any of the compromised systems. At a minimum, the attacker would need to specify how long to record as well as file storage and network transmission options. In our implementation the attacker can specify: the UDP server listening port number; how long to record for; whether to use a file, network stream, or both; the output filename; and, the network broadcast stream destination host IP address and port number. For run time controls, the attacker can send commands to the bug program through its UDP server.

4 Detection and Mitigation

Limiting microphone access can be done either in hardware, such as with a physical kill switch or cover, or in software like other resource controls such as application firewalls that monitor network access. Physical switches would be a difficult after-market option, and unlike application firewalls which have large market acceptance there appears to be no existing generic software based protection against microphone surveillance attacks.

There are particular reasons why monitoring microphone access should be a low burden to the user. First, unlike network access requests or prompts for privilege escalation the average low-tech user is capable of understanding the purpose of the microphone and when it should be turned on or off. Second, the frequency of microphone access requests should be much lower than other resources making it easier to track which applications should be permitted. This also prevents illegitimate access requests from hiding in a cluster of legitimate requests.

In this section we present a preliminary detection and mitigation mechanism for threats similar to bugbot. Our application can can detect if a Windows process is actively using the microphone and allow the user to set access controls similar to antivirus suites and application firewalls based on API call monitoring. This type of specification based intrusion detection [16] is accomplished by injecting target processes with a custom dynamic-linked library (DLL) that sets wrappers, known as hooks, for Windows API (WinAPI) calls.

When the monitored process calls a hooked function the injected DLL's version of the function is used instead. This then provides the DLL with transparent access to all arguments and the ability to return arbitrary values. For this paper, we used a free Microsoft Research package titled Detours [14] that provides tools and a simplified API for coding wrapper DLLs.

In section 4.1 we detail how to completely deny suspect processes. While this provides a solution that protects the true audio, it reduces the chances of tracing the source of the attack. In section 4.2 we present a solution to this problem by demonstrating how the victim can deceive the attacker by providing a decoy sound. The final product is tested section 4.3 where we present several scenarios and results.

4.1 Deploying the Protection Mechanism

As described in section 2.1, this demonstration uses the waveIn WinAPI and details are in terms of those functions. It should be noted that if other WinAPI functions are used, the same concept could be executed but with different functions detoured. In the bug program there are two pertinent function calls that are candidates for hooking into. A detour of `waveInOpen` would interfere with passing initialization data to the sound card driver, but a more direct way to intervene would be to hook into `waveInStart`. Once the DLL has detoured the bug's call it has the option to prevent the bug process from calling the true `waveInStart` function and return a

failure value instead. This is an optimal place to insert an allow-or-deny behavior since a denied bug would simply fail to reach a state capable of gathering data.

Automating the decision of whether a process should be trusted or untrusted is a difficult problem. A simple and reliable technique, as we have implemented, is for the monitoring DLL to prompt the user to approve microphone requests on a case to case basis. It is safe to assume that while allow or deny decisions for frequently requested resources such as outbound network access can easily confuse untrained users, most know when they are or are not using their microphone. However, since a denied bug would be obvious to the attacker, a more effective response may be through misinformation, as we present in the next section.

4.2 Deception by Decoy Audio

In cases where it is necessary to have an audit trail, or there is a desire to fully trace an attack, it is advisable to create as much time between detecting an intrusion and the remote attacker leaving the system. One way is to deceive the attacker by feeding the bug application crafted data. This method maintains viability even if a future surveillance program uses a yet undiscovered covert channel for exporting the data. This decoy sound should be believable and unpredictable so as to remain undiscovered in the hopes of buying enough time to permit a better trace to the source of the attack. For example, randomized keyboard clacking or indiscernible background mumble would be good candidates.

While the complete traceback of the remote attacker is still an open research problem, this technique is a building block toward such a goal. With properly crafted decoy audio, such as timed silence between predictable sounds, it is possible to introduce distinguishable elements into the transmitted data stream. Similar traceback methods have been used for other applications [33]. This could even hold in the case of compression or encryption, as recent research [25, 36] has illustrated strong correlations between such streams and their original content.

To accomplish the deception method transparently, the Detours DLL should inject the crafted audio by replacing filled buffers, that the sound card returns, before the bug can read them. As illustrated in figure 4, if the DLL hooks into the WinAPI `GetMessage` call (1), then it could intercept (1*) the message from sound card driver (5) indicating a filled buffer. Inside of this message is a pointer to the buffer and the size of the data stored in the buffer. If the DLL also detoured the `waveInOpen` call (2), then it would know the format of the raw data in the buffer in order to match the decoy audio with it. This avoids deciphering the bug application's internal data structures and formats for storing the data. At this point the DLL could swap (5*) the buffer for an equal length snip from the decoy audio.

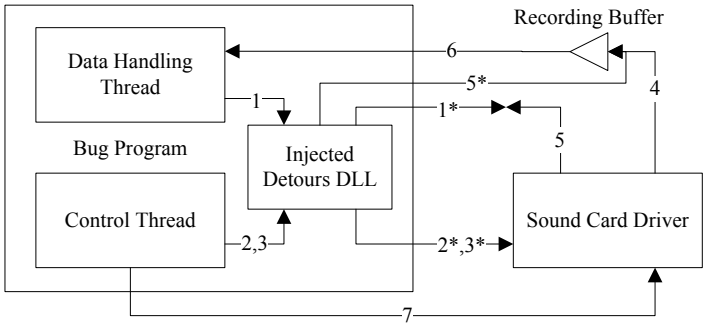


Fig. 4 Visualization of the bug program control flow using the deception method. Numbers refer to same steps as figure 2. 1*, 2*, and 3* are 1, 2, and 3 after DLL interception. 5*) After the DLL receives a "full buffer" message, it replaces the recording buffer with decoy audio and then passes the message to the data handling thread.

4.3 Test Scenarios

In order to observe the viability of injected DLLs that monitor processes for microphone requests we implemented the defense as a single DLL. When the monitor catches a request it prompts the user to either allow, deny, or deceive, by means of decoy audio, the process in question. In order to establish a reliable testing environment we ran two instances of the bugbot program, one to represent the untrusted bugbot process and the other to represent an arbitrary trusted processes.

We initially examined baseline tests with the bugbot and trusted application running separately. When either was not monitored, or monitored but allowed by the user, it had access to the true audio. When either was monitored and denied, or deceived, then neither could access the microphone's audio.

In the next tests we concurrently executed monitored instances of the applications to demonstrate that a monitor in deception mode would not interfere with legitimate recordings. We ran two sets of tests, in one the bugbot attempted to record before the trusted application initialized the microphone and in the other the bugbot attempted to record after. In both cases, when the user chose to deceive the bugbot our mitigation technique transparently replaced the audio from the microphone with a specified recording loop. As a result, the attacker heard the decoy sound while the trusted application continued using the true microphone input.

5 Related Works

Malware detection has been an area of active research, and there are many methods proposed to detect or mitigate malware [7–9, 12, 13, 27, 32, 35]. StackGuard [6], StackGhost [10], RAD [4] and Windows vaccination [21] prevent stack based over-

flow by protecting the return address from being modified by the malware. However, they are not effective against other attack vectors such as heap based overflow [5].

Another method, packet vaccine [34], seeks to detect malware exploit packets by randomizing address-like stings in the packet payloads. Similar to other randomization based approaches [1, 2, 17, 18, 24], which protect applications and systems via randomizing the instruction set or address layout, packet vaccine will cause the vulnerable applications to crash when they are exploited by malware.

Taint analysis aims to detect illegal information flow by tracking the taint, and it has been widely used for analyzing malware [3,22,26,28,29,31,37]. As pointed out by Saxena et al. [26], taint tracking usually incurs high performance overhead. This makes it difficult to be used for detecting malware in real-time.

To the best of our knowledge, no existing malware defense approach has been shown to be effective in detecting the bugbot we have presented.

6 Conclusion

Remote surveillance is a significantly invasive threat, arguably even more so than identity theft. As it stands now, most vulnerable devices do not have the protection necessary to distinctly address microphone or camera hijacks. As a growing number of mobile devices with exploitable operation systems gain more reliable Internet access, this long standing problem is reaching a critical potential.

The risk of surveillance attacks is increased on systems shared with untrusted users. Since multiple users can open the microphone simultaneously, regardless of who is physically at the system, any user of a system can be compromised even if just one user of that system is not protected. Imagine a spouse that exploits this weakness on purpose to spy on his or her partner through a shared computer. This leads to questioning how to properly handle the lack of control over shared resources as more people adopt true multi-user environments.

To demonstrate the viability of a surveillance intrusion, we developed a modern interpretation of a stealthy microphone hijack threat. The features of the bugnet closely match in-the-wild exploits. It uses a botnet framework and is able to exploit a system as soon as the target connects to the Internet.

We then investigated ways to mitigate the threat. Physical protection is an option, such as a cover or on-off switch, but most devices do not have this built-in, leaving software as the only answer for a vast majority of the vulnerable systems. Given the infrequency of microphone access by the average user, adding a way to monitor and interactively control recording access should be unobtrusive. As a solution we developed a mitigation mechanism that can be broadly applied to detect and prevent surveillance exploits. This methodology employs API hooks to monitor processes and uses extensible permissions testing to provide an allow-or-deny behavior.

To facilitate forensic analysis, our bugbot mitigation technique additionally involves using a decoy audio loop that consists of well crafted believable noise, such as background keyboard clacking or indiscernible talking to retain the remote at-

tacker's network connection while keeping the true audio recording confidential. The additional time created could then be used to trace the source of the attacker's connection, or at minimum, gathering as much audit information as possible.

Currently most devices with network access and microphones, such as laptops and smartphones, are vulnerable to this type of attack. Yet there is still no widely accepted way for users to protect themselves. As awareness of this problem increases, the potential threat to privacy may lead consumers and businesses to lessen their dependence on such devices.

References

1. Barrantes, E., Ackley, D., Forrest, S., Palmer, T., Stefanovic, D., Zovi, D.: Randomized instruction set emulation to disrupt binary code injection attacks. In: Proc. of the 10th ACM Conf. on Computer and Communications Security (CCS 2003), pp. 281–289. ACM (2003)
2. Bhatkar, S., DuVarney, D.C., Sekar, R.: Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In: Proc. of the 12th USENIX Security Sym. (2003)
3. Chen, S., Xu, J., Nakka, N., Kalbarczyk, Z., Iyer, R.K.: Defeating memory corruption attacks via pointer taintedness detection. In: Proc. of the 2005 International Conf. on Dependable Systems and Networks (DSN 2005). IEEE (2005)
4. Chiueh, T., Hsu, F.H.: RAD: A compile-time solution to buffer overflow attacks. In: Proc. of the 21st International Conf. on Distributed Computing Systems (ICDCS 2001), pp. 409–417. IEEE (2001)
5. Conover, M.: w00w00 on heap overflows (1999). `http://www.w00w00.org/files/articles/heaptut.txt`
6. Cowan, C., Pu, C., Maier, D., Hinton, H., Walpole, J., Bakke, P., Beattie, S., Grier, A., Wagle, P., Zhang, Q.: StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In: Proc. of the 7th USENIX Security Sym., pp. 63–78 (1998)
7. Feng, H.H., Giffin, J.T., Huang, Y., Jha, S., Lee, W., Miller, B.P.: Formalizing sensitivity in static analysis for intrusion detection. In: Proc. of the 2004 IEEE Sym. on Security and Privacy (S&P 2004) (2004)
8. Feng, H.H., Kolesnikov, O.M., Fogla, P., Lee, W., Gong, W.: Anomaly detection using call stack information. In: Proc. of the 2003 IEEE Sym. on Security and Privacy (S&P 2003) (2003)
9. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A sense of self for unix processes. In: Proc. of the 1996 IEEE Sym. on Security and Privacy (S&P 1996). IEEE (1996)
10. Frantzen, M., Shuey, M.: StackGhost: Hardware facilitated stack protection. In: Proc. of the 10th USENIX Security Sym., pp. 55–66 (2001)
11. Gibson, S.: Spyware was inevitable. Commun. ACM **48**(8), 37–39 (2005)
12. Giffin, J.T., Dagon, D., Jha, S., Lee, W., Miller, B.P.: Environment-sensitive intrusion detection. In: Proc. of the 8th International Sym. on Recent Advances in Intrusion Detection (RAID 2005) (2005)
13. Giffin, J.T., Jha, S., Miller, B.P.: Efficient context-sensitive intrusion detection. In: Proc. of the 11th Network and Distributed System Security Sym. (NDSS 2004) (2004)
14. Hunt, G., Brubacher, D.: Detours: Binary interception of Win32 functions. In: Proc. of the 3rd USENIX Windows NT Sym., pp. 135–143 (1999)
15. Ianelli, N., Hackworth, A.: Botnets as a vehicle for online crime. Tech. rep., CERT (2005)
16. Idika, N., Mathur, A.P.: A survey of malware detection techniques. Tech. rep., SERC (2007). SERC-TR-286
17. Jun Xu, Z.K., Iyer, R.K.: Transparent runtime randomization for security. In: Proc. of the 22nd Sym. on Reliable and Distributed Systems (SRDS 2003). IEEE (2003)

18. Kc, G.S., Keromytis, A.D., Prevelakis, V.: Countering code-injection attacks with instruction-set randomization. In: Proc. of the 10th ACM Conf. on Computer and Communications Security (CCS 2003), pp. 272–280. ACM (2003)
19. Leavitt, N.: Mobile phones: the next frontier for hackers? IEEE Computer **38**(4), 20–23 (2005). DOI 10.1109/MC.2005.134
20. McCullagh, D.: FBI taps cell phone mic as eavesdropping tool. ZDNet News (2006)
21. Nebenzahl, D., Sagiv, M., Wool, A.: Install-time vaccination of Windows executables to defend against stack smashing attacks. IEEE Transactions on Dependable and Secure Computing (TDSC) **3**(1), 78–90 (2006)
22. Newsome, J., Song, D.: Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In: Proc. of the 12th Network and Distributed System Security Sym. (NDSS 2005) (2005)
23. Rajab, M.A., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: IMC '06: Proc. of the 6th ACM SIGCOMM Conf. on Internet Measurement (2006)
24. Sandeep Bhatkar, R.S., DuVarney, D.C.: Efficient techniques for comprehensive protection from memory error exploits. In: Proc. of the 14th USENIX Security Sym. (2005)
25. Saponas, S.T., Lester, J., Hartung, C., Agarwal, S., Kohno, T.: Devices that tell on you: Privacy trends in consumer ubiquitous computing. In: Proc. of the 16th USENIX Security Sym., pp. 55–70 (2007)
26. Saxena, P., Sekar, R., Puranik, V.: Efficient fine-grained binary instrumentation with applications to taint-tracking. In: Proc. of the 2008 International Sym. on Code Generation and Optimization (CGO 2008) (2008)
27. Sekar, R., Bendre, M., Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In: Proc. of the 2001 IEEE Sym. on Security and Privacy (S&P 2001) (2001)
28. Shankar, U., Talwar, K., Foster, J.S., Wagner, D.: Detecting format string vulnerabilities with type qualifiers. In: Proc. of the 10th USENIX Security Sym. (2001)
29. Su, Z., Wassermann, G.: The essence of command injection attacks in web applications. In: Proc. of the 33rd ACM Sym. on Principles of Programming Languages (POPL 2006) (2006)
30. Trilling, S., Nachenberg, C.: The future of malware. In: EICAR 1999 best paper proceedings (1999)
31. Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C., Vigna, G.: Cross site scripting prevention with dynamic data tainting and static analysis. In: Proc. of the 14th Network and Distributed System Security Sym. (NDSS 2007) (2007)
32. Wagner, D., Dean, D.: Intrusion detection via static analysis. In: Proc. of the 2001 IEEE Sym. on Security and Privacy (S&P 2001) (2001)
33. Wang, X., Chen, S., Jajodia, S.: Tracking anonymous peer-to-peer VoIP calls on the internet. In: Proc. of the 12th ACM Conf. on Computer Communications Security (2005)
34. Wang, X., Li, Z., Xu, J., Reiter, M.K., Kil, C., Choi, J.Y.: Packet vaccine: Black-box exploit detection and signature generation. In: Proc. of the 13th ACM Conf. on Computer and Communications Security (CCS 2006). ACM (2006)
35. Warrender, C., Forrest, S., Pearlmutter, B.: Detecting intrusions using system calls: Alternative data models. In: Proc. of the 1999 IEEE Sym. on Security and Privacy (S&P 1999), pp. 133–145 (1999)
36. Wright, C.V., Ballard, L., Monrose, F., Masson, G.M.: Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob? In: Proc. of the 16th USENIX Security Sym. (2007)
37. Xu, W., Bhatkar, S., Sekar, R.: Taint-enhanced policy enforcement: A practical approach to defeat a wide range of attacks. In: Proc. of the 15th USENIX Security Sym. (2006)
38. Zaystev, O.: Rootkits, Spyware/Adware, Keyloggers and Backdoors: Detection and Neutralization. A-List Publishing (2006)