

# Autonomix: Component, Network, and Systemic Autonomy: Survivability Validation Characterization

Crispin Cowan  
WireX Communications, Inc.  
<http://immunix.org/autonomix/>

## 1 Technology Description and Information Assurance/ Survivability Problem Addressed

The Autonomix view of system survivability is to make systems survive attempts by attackers to exploit unknown vulnerabilities in system software. We approach this problem by employing the detect-respond-adapt loop. We sub-divide the problem by approaching this loop at three different levels of granularity: Component Autonomy, Network Autonomy, and System Autonomy. Throughout this report, we will provide subsections that describe each of Component, Network, and Systemic Autonomy separately.

### 1.1 Component Autonomy

These techniques provide individual components with completely autonomous defenses. Attacks are detected and responded to within the component, e.g. the StackGuard defense. Component Autonomy technologies being developed include:

**PointGuard:** A generalization of StackGuard [4, 3] that generalizes StackGuard's protection to all pointers in a program.

**RaceGuard:** A systematic method to protect against file system race conditions, the second most common form of system vulnerability [6].

**IP Guard:** A systematic method to defend kernels against network packet sequence inconsistency DoS attacks such as Teardrop and Land.

**CryptoMark:** A kernel enhancement and set of utilities to cryptographically sign executable programs, and enforce that only signed programs may run.

**FormatGuard:** a compiler approach to defend against the (now common) problem of `printf` format string vulnerabilities [2].

**LSM (Linux Security Module):** a community-based project to build a standard interface for the mainline Linux kernel that enables the construction of security-enhancing loadable kernel modules.

## 1.2 Network Autonomy

By providing standard Internet protocol gateways such as XML, LDAP, NIS, and SNMP to the OASIS family of adaptive response systems, we leverage OASIS results by expanding the scope of system components that can detect, reason about, and respond to attacks. The Autonomix proposal envisioned Network Autonomy as a gateway between CIDF events and assorted standard network protocols such as XML, SNMP, LDAP, and NIS. Since then, CIDF has been de-emphasized, leaving Network Autonomy with a lack of focus.

Concurrent with this change of emphasis, the OGI portion of the Autonomix team (Lois Delcambre and Shawn Bowers) developed a *meta-modeling system* for representing and transforming model-based information [1] such as intrusion detection events. We have used this tool to develop some filters for transforming intrusion events from one model to another:

**cidf2xml:** Converts plain-text GIDOs (as messages) to XML. The GIDOs are assumed to be in the format used by Boeing's Convert tool, which produces plain-text GIDO messages from binary GIDOs. Additionally, *cidf2xml* can be run in reverse, which converts the XML GIDO messages back into plain-text GIDO messages. Note that the entire CISL language is not recognized at this time by *cidf2xml*. Only the portion of the language used in Boeing's example files are accounted for.

**idmef2cisl:** Takes an IDMEF XML file and converts it into a CISL plain-text GIDO message. *idmef2cisl* uses both the XMLExtractor and the *cidf2xml* tools. *Idmef2cisl* leverages a fixed-mapping that is appropriate for the example IDMEF alerts presented in the IDMEF specification. There are a number of potential ways to map between IDMEF and CISL, however, this tool uses a single, fixed mapping to show proof-of-concept.

## 1.3 Systemic Autonomy

By providing a formal method of reasoning about alternative system configurations, we enrich the assurance that a site administrator can gain from employing autonomously adaptive components. A formal basis for understanding the impact of adaptive responses enables the deployment of more complex automatic responses to intrusions.

Systemic Autonomy uses *Adaptation Space* (developed under “Heterodyne” DARPA grant F30602-96-1-0302) as a formalism for reasoning about “alternative implementations” of a system (“orchestration”, in the parlance of the recent AIA meeting). We use and extend adaptation spaces to model the adaptive responses employed by an *autonomic information assurance* system. “Alternative implementations” refers to any kind of reconfiguration, ranging from changing a single bit of state to a complete replacement of the implementing software and hardware. The purpose of an adaptation space is to enable formal reasoning about which alternative implementation is to be used under a given set of circumstances.

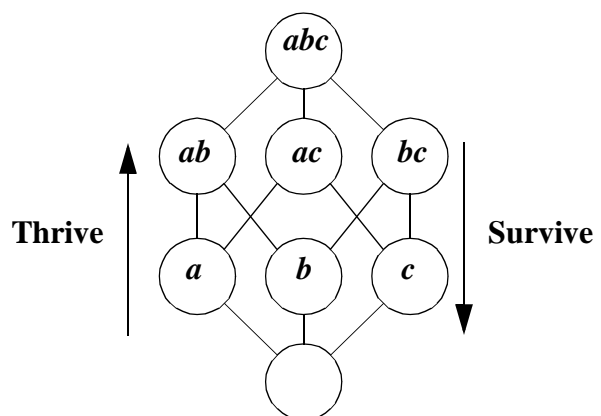
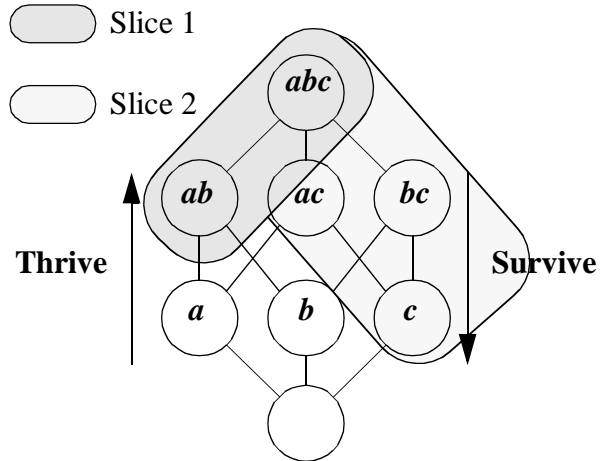


Figure 1 Simple Adaptation Space

Adaptation space functions by selecting among alternative implementations according to *preference* and *current conditions*. Figure 1 shows a sample adaptation space describing three conditions  $a$ ,  $b$ , and  $c$ . The top node is where  $a$ ,  $b$ , and  $c$  hold, while the bottom node is where none of the conditions hold. *Downward adaptations* switch to an implementation that can function correctly with fewer assumptions (i.e. fewer true conditions) about the execution environment, and are said to be *surviving adaptations*. *Upward adaptations* switch to implementations that require more assumptions about the execution environment to hold, but also provide additional functionality or quality of service, and are said to be *thriving adaptations*.



**Figure 2 Adaptation Space Sliced into Alternative Implementations**

## 2 Assumptions

### 2.1 Component Autonomy

**AC1:** that most vulnerabilities fall within certain classes (buffer overflows, race conditions, printf format bugs, etc.)

**AC2:** that exploitation attempts within these classes of vulnerabilities can be detected in real time, with low false negatives, and (nearly) zero false positives, enabling intrusion detection to become intrusion *rejection*.

**AC3:** that source code for most of the vulnerable software of interest is readily available.

### 2.2 Network Autonomy

**AN1:** that there exist some non-Autonomix technologies that do effective intrusion detection and response.

**AN2:** that these intrusion detection and response components want to communicate.

**AN3:** that there is a diversity of languages or protocols for communicating intrusion events and responses.

**AN4:** that these languages and protocols are similar enough to be mechanically translated without unacceptable loss of information.

**AN5:** that these languages and protocols are different enough that mechanical translation requires more sophistication than simple string substitution.

### 2.3 Systemic Autonomy

**AS1:** that there exist *separate* intrusion detection and response technologies that need to be organized.

**AS2:** that there exist effective intrusion response strategies.

**AS3:** that these intrusion response strategies can be characterized as distinct *alternative implementations*.

**AS4:** that the appropriate time to deploy these *alternative implementations* can be characterized in terms of *use conditions*, e.g. the condition that the site firewall has been port-scanned within the last hour, or that the user jbløe has had three sequential password failures.

## 3 Residual Risks, Limitations, and Caveats

### 3.1 Component Autonomy

**RC1:** vulnerabilities that do not fit into easily classifiable classes of software defects, e.g. “stupid programmer error” or “generic logic flaw.”

**RC2:** vulnerabilities that are classifiable, but are not addressed by Component Autonomy tools.

**RC3:** vulnerabilities due to mis-design, rather than mis-implementation.

**RC4:** vulnerabilities due to mis-configuration, rather than mis-implementation.

### 3.2 Network Autonomy

**RN1:** loss of information or precision in intrusion due to mis-translation.

**RN2:** intrusion response arriving too late to be effective due to latency induced by protocol translation.

### 3.3 Systemic Autonomy

**RS1:** non-existence or limited range of effective responses to intrusion.

## 4 Vulnerabilities and Attacks

### 4.1 Component Autonomy

Component Autonomy largely defends against software defects that occur at *implementation* time. These can be viewed as design-time defects of the C programming language and its libraries (e.g. buffer overflows, format bugs, and temporary file race conditions) but because Autonomix seeks to enhance the survivability of *existing* software, changing the C language and its libraries is not a feasible solution.

The exception to this rule is LSM, which is intended to facilitate the insertion of access control modules into the standard Linux kernel. Access control modules provide for different models and granularities of access control, which generally address design issues with the security permissions model provided by the standard Linux kernel.

The *classes* of vulnerabilities that Component Autonomy seeks to address are as follows:

**AV1:** stack smashes, i.e. buffer overflow attacks intended to corrupt activation records.

- AV2:** stack overflows, i.e. buffer overflows of stack-allocated strings intended to corrupt other stack-allocated objects.
- AV3:** heap overflows, i.e. buffer overflows of strings allocated on the heap (malloc()) to corrupt adjacent heap objects.
- AV4:** static overflows, i.e. buffer overflows of statically allocated strings to corrupt adjacent statically allocated objects.
- AV5:** printf format string bugs involving the standard printf family of functions in libc.
- AV6:** printf format string bugs involving other printf-like functions.
- AV7:** temporary file race conditions.
- AV8:** other, non-race issues with temporary files.
- AV9:** other, non-temporary file issues that are race conditions.
- AV10:** network corruption attacks, where the attacker sends a creatively invalid sequence of datagrams to the victim machine with the intent of crashing the victim's network protocol stack.
- AV11:** malicious code insertion, where the attacker installs malicious code on the victim's machine, such as Trojan Horse programs, password sniffers, or DDoS zombies.
- AV12:** abuse of authority, where attackers mis-use authority mistakenly or intentionally granted to them because of limitations in the (Linux) host operating system's access control mechanisms.

## 4.2 Network Autonomy

Network Autonomy is intended to enhance the effectiveness of other systems' intrusion detection and intrusion response capabilities. As such, Network Autonomy is non-specific in the vulnerabilities and attacks that it addresses.

## 4.3 Systemic Autonomy

Systemic Autonomy is also intended to enhance the effectiveness of other systems' intrusion detection and intrusion response capabilities. As such, Systemic Autonomy is also non-specific in the vulnerabilities and attacks that it addresses.

# 5 Information Assurance and Survivability Attributes

## 5.1 Component Autonomy

Component Autonomy primarily defends the *integrity* (I) of the computing systems themselves, by preventing the attacker from corrupting the behavior of the system using any of the common methods of software attack. By protecting system and application software from common modes of corruption, Component Autonomy *indirectly* defends all of the other survivability attributes: *availability* (AV, the attacker cannot take down services by corrupting them), *confidentiality* (C, the attacker cannot extract secrets by coercing a program into disclosing the secret), *authentication* (AU, the attacker cannot spoof authentication by corrupting the authentication software), and *nonrepu-*

*diation* (NR, a marginal ability to ensure nonrepudiation by preventing the attacker from Trojaning the document authenticator).

## 5.2 Network Autonomy

Network Autonomy is intended to enhance the effectiveness of other systems' intrusion detection and intrusion response capabilities. As such, Network Autonomy is non-specific in the survivability attributes that it addresses.

## 5.3 Systemic Autonomy

Systemic Autonomy is intended to enhance the effectiveness of other systems' intrusion detection and intrusion response capabilities. As such, Systemic Autonomy is non-specific in the survivability attributes that it addresses.

# 6 Comparison with Other Systems

## 6.1 Component Autonomy

**PointGuard:** The only significant competitor to PointGuard is the Bounded Pointers project [9] which seeks to add full array bounds checking to the GNU C Compiler. The expected result should be that Bounded Pointers will provide broader security coverage, while PointGuard will provide lower overhead. Bounded Pointers is not a commercial product, and is a work in progress.

**RaceGuard:** The major competitor for RaceGuard is part of the security enhancement package for Linux known as Open Wall [8]. Rather than attacking the "race" aspect, this enhancement to the Linux kernel attacks the propensity for privileged programs to create and follow links. Two restrictions are imposed: processes will not follow links in directories with the +t ("sticky") bit set unless the owner is trusted (same UID as the process, or owns the directory), and processes are not allowed to create hard links to files that they do not own. OpenWall is not a commercial product.

**IP Guard:** IPGuard has no known competition, other than due diligence in implementing network protocol software.

**CryptoMark:** The CryptoMark competitors are:

**Arbaugh et al:** A close competitor to CryptoMark is a research prototype by Arbaugh et al [11] that provides substantially similar functionality.

**SecureEXE:** SecureEXE from Securewave.com provides similar functionality for Windows.

**Tripwire:** Tripwire is loosely related to CryptoMark, providing broader security coverage (monitoring both program and data files) but providing only intrusion detection (not rejection) and imposing considerable overhead in the form of many false positive reports.

**FormatGuard:** FormatGuard competitors all come in the form of static analysis tools that attempt to detect printf format bugs. Static analysis tools have the advantage of stopping bad code from shipping, at the expense of less precise detection: all of these tools exhibit both false negatives

(missed format bugs) and many exhibit false positives (reporting instances that are not actually format bugs). The competing tools are:

**Wagner et al:** built an extension to C and an analysis tool that allows the programmer to specify data as “tainted” when it comes direct from potential attackers, and the analyzer follows the data flow to detect cases where the attacker can provide the format string. Unlike the other technologies, this tool requires active intervention on the part of the programmer to make the tool effective [10].

**PScan:** scans source code looking for printf calls in which the last argument is the format string, and it is static [7].

**GCC:** the command line option “-Wformat=2” will cause GCC to complain about non-static format strings. This generates too many false positives, so an enhancement has been developed that restricts complaints to the special case of “printf(foo)” which is the common case for format bugs.

**LSM (Linux Security Module):** has no competitors. LSM is a community effort, designed to enable competing security modules to all be able to load into the standard Linux kernel. Competition is intended to happen *within* the LSM community.

## 6.2 Network Autonomy

Network Autonomy seeks to translate between competing standards for intrusion detection and response events. The main competitor to this approach is the standardization of intrusion detection and response events, in the form of the IETF IDWG (Intrusion Detection Working Group). If successful, the IDWG standard(s) will eliminate the need for Network Autonomy.

## 6.3 Systemic Autonomy

Systemic Autonomy is an “orchestrator” from the AIA experiment, intended to select appropriate responses to a detected intrusion attempt. There are a variety of competitors from the AIA program; the one being used in the SARA experiment is the “SoSMART” orchestrator. The main problem for orchestrators is the relative dearth of effective responses to intrusion.

# 7 Information Assurance and Survivability Mechanisms

## 7.1 Component Autonomy

**M0: StackGuard:** not an Autonomix project, but rather built by Immunix, the ancestor to Autonomix [4, 3, 5]. We include it for the Rationale Matrix in Section 8.

**M1: PointGuard:** A generalization of StackGuard that generalizes StackGuard’s protection to all pointers in a program.

**M2: RaceGuard:** A systematic method to protect against file system race conditions, the second most common form of system vulnerability [6].

**M3: IP Guard:** A systematic method to defend kernels against network packet sequence inconsistency DoS attacks such as Teardrop and Land.

**M4: CryptoMark:** A kernel enhancement and set of utilities to cryptographically sign executable programs, and enforce that only signed programs may run.

**M5: FormatGuard:** a compiler approach to defend against the (now common) problem of `printf` format string vulnerabilities [2].

**M6: LSM (Linux Security Module):** a community-based project to build a standard interface for the mainline Linux kernel that enables the construction of security-enhancing loadable kernel modules.

## 7.2 Network Autonomy

Network Autonomy provides a *meta-modeling system* for representing and transforming model-based information [1] such as intrusion detection events. We have used this tool to develop some filters for transforming intrusion events from one model to another:

**M7: cidf2xml:** Converts plain-text GIDOs (as messages) to XML. The GIDOs are assumed to be in the format used by Boeing's Convert tool, which produces plain-text GIDO messages from binary GIDOs. Additionally, `cidf2xml` can be run in reverse, which converts the XML GIDO messages back into plain-text GIDO messages. Note that the entire CISL language is not recognized at this time by `cidf2xml`. Only the portion of the language used in Boeing's example files are accounted for.

**M8 idmef2cisl:** Takes an IDMEF XML file and converts it into a CISL plain-text GIDO message. `idmef2cisl` uses both the `XMLExtractor` and the `cidf2xml` tools. `Idmef2cisl` leverages a fixed-mapping that is appropriate for the example IDMEF alerts presented in the IDMEF specification. There are a number of potential ways to map between IDMEF and CISL, however, this tool uses a single, fixed mapping to show proof-of-concept.

## 7.3 Systemic Autonomy

**M9: Adaptation Space Navigator:** A Java program that loads in a description of an adaptation space (encoded in XML), takes current *use conditions* as input, and outputs the current preferred *alternative implementation*.

## 8 Rationale

**Table 1: Implementation Time Issues**

Vulnerability	AV	I	C	AU	NR
AV1		M0, M1 <sup>a</sup>			
AV2		M1 <sup>b</sup>			
AV3		M1 <sup>c</sup>			
AV4		M1 <sup>d</sup>			
AV5		M1, M5 <sup>e</sup>			
AV6		M1 <sup>f</sup>			
AV7		M2 <sup>g</sup>			
AV10	M3 <sup>h</sup>				

a.StackGuard (M0) is specifically designed to stop AV1 (Stack smash), while PointGuard (M1) generically stops corruption of pointers via buffer overflows.

b.PointGuard is specifically designed to stop this attack.

c.PointGuard is specifically designed to stop this attack.

d.PointGuard is specifically designed to stop this attack.

e.FormatGuard is specifically designed to stop this attack. PointGuard stops it as a side-effect of encrypting pointers.

f.PointGuard stops it as a side-effect of encrypting pointers, which is valuable because this class escapes FormatGuard protection.

g.RaceGuard is specifically designed to stop this attack.

h.IPGuard is specifically designed to stop this attack.

**Table 2: Operation Time Issues**

Vulnerability	AV	I	C	AU	NR
AV8		M6 <sup>a</sup>			
AV9		M6 <sup>b</sup>			
AV11		M4 <sup>c</sup>	M4	M4	M4
AV12	M6 <sup>d</sup>	M6	M6	M6	M6

a.Non-race issues with temporary files is a specific case of the general problem of poor file permissions management. It is hoped that superior access control management in the form of an LSM access control module would address this problem.

b.It is similarly hoped that an access control module might address non-file race vulnerabilities, but this is a *faint* hope.

c.M4 (CryptoMark) is specifically designed to address AV11 (malicious code insertion), which potentially affects integrity, confidentiality, authentication, and nonrepudiation.

d.M6 (LSM) enables the creation of arbitrary access control modules for the Linux kernel. As such, it can potentially affect all of the survivability attributes.

## 9 Cost and Benefit Analysis

None of the Autonomix tools presented impose significant storage costs. Here we present the costs and benefits that are germane to the tools.

**Table 3: Costs & Benefits**

Tool	Performance Cost	Compatibility Cost	False Positives	Relative Invulnerability
M0: StackGuard	nil	2/500 required modification to work	nil	6/57 of the Red Hat Linux 7.0 vulnerabilities
M1: PointGuard	work in progress; estimate less than 20% overhead	work in progress	work in progress	work in progress
M2: RaceGuard	less than 1%	nil	nil	21/57
M3: IPGuard	work in progress	work in progress	work in progress	work in progress
M4: CryptoMark	work in progress	work in progress	work in progress	work in progress
M5: FormatGuard	less than 1%	70/500 programs required trivial modification	nil	6/57
M6: LSM	less than 2%	depends on module used	depends on module used	work in progress
Combination of M0, M2, and M5	less than 2%	72/500 programs required modifications to work	nil	33/57

## References

- [1] Shawn Bowers and Lois Delcambre. Representing and Transforming Model-Based Information. In *Workshop on Semantic Web: Models, Architectures and Management*, Lisbon, Spain, September 2000. Held in conjunction with the European Conference on Digital Libraries.
- [2] Crispin Cowan, Matt Barringer, Steve Beattie, Greg Kroah-Hartman, Mike Frantzen, and Jamie Lokier. FormatGuard: Automatic Protection From printf Format String Vulnerabilities. In *USENIX Security Symposium*, Washington, DC, August 2001.
- [3] Crispin Cowan, Steve Beattie, Ryan Finnin Day, Calton Pu, Perry Wagle, and Erik Walthinsen. Protecting Systems from Stack Smashing Attacks with StackGuard. In *Linux*

*Expo*, Raleigh, NC, May 1999.

- [4] Crispin Cowan, Calton Pu, Dave Maier, Heather Hinton, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, and Qian Zhang. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *7th USENIX Security Conference*, pages 63–77, San Antonio, TX, January 1998.
- [5] Crispin Cowan, Perry Wagle, Calton Pu, Steve Beattie, and Jonathan Walpole. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. In *DARPA Information Survivability Conference and Expo (DISCEX)*, January 2000. Also presented as an invited talk at SANS 2000, March 23-26, 2000, Orlando, FL, <http://schafercorp-ballston.com/discex>.
- [6] Crispin Cowan, Steve Beattie Chris Wright, and Greg Kroah-Hartman. RaceGuard: Kernel Protection From Temporary File Race Vulnerabilities. In *USENIX Security Symposium*, Washington, DC, August 2001.
- [7] Alan DeKok. PScan: A limited problem scanner for C source files. Bugtraq mailing list, <http://www.securityfocus.com/archive/1/68688> and the web <http://www.striker.ottawa.on.ca/~aland/pscan/>, July 7 2000.
- [8] “Solar Designer”. Root Programs and Links. <http://www.openwall.com/linux/>.
- [9] Greg McGary. Bounds Checking in C & C++ Using Bounded Pointers. <http://gcc.gnu.org/projects/bp/main.html>, 2000.
- [10] Umesh Shankar, Kunal Talwar, Jeff Foster, and David Wagner. Automated Detection of Format-String Vulnerabilities. In *USENIX Security Symposium*, Washington, DC, August 2001.
- [11] Leendert van Doorn, Grerco Ballintijn, and William A. Arbaugh. Signed Executables for Linux. Technical Report UMD CS-TR-4259, University of Maryland, 2001.